Using Repositories for Ontology Design and Semantic Mapping

by

Ali Hashemi

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Using Repositories for Ontology Design and Semantic Mapping

Ali Hashemi
MASc
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2009

There are two significant impedances to the realization of the potential of ontologies. First, many ontology designers lack the necessary background in formal logics to express their intuitions clearly and precisely, resulting in the proliferation of ontologies with low expressivity. Concurrently, developing semantic mappings between existing ontologies is difficult, because much of the semantics is external to the representation. This thesis uses the idea of metaphor to develop architectures for ontology repositories to serve as bottom-up reusable resources. Moreover, an ontology design algorithm has been developed that allows designers to communicate their ideas at the semantic level, simply by generating and vetting models. Finally, a semantic mapping algorithm has been developed that uses an ontology repository to determine the *similarities* and *differences* between any number of target ontologies. An ontology for partial orders has been elaborated to demonstrate the proof of concept and populate the first iteration of the repository.

Acknowledgements

This thesis has been the culmination of the perpetual casting of wide nets followed by the slow filtering and synthesis of the catch into a (hopefully) useful and interesting outcome.

To that end, it has been a privilege and pleasure to work with Michael Gruninger. He has been a tremendous help in focusing my broad interests to a fruitful engineering endeavor. His insightful comments and feedback are valued tremendously. One should consider themselves lucky to have them as an advisor.

Secondly, I would like to thank my good friend Stephen Kershaw for indulging my thoughts and listening on many a night to musings on philosophy, logic, communication and a wide array of subjects.

I would also like to thank my mother, father and sister for all that they have done and their tremendous support, without them, not much would have been possible.

The questions and feedback from my thesis committee members, Mark Fox and Gary Bader were a tremendous help.

Laslty, I would like to thank my friends and labmates who helped me remain grounded in the things that matter. It's been a pleasure. Thank you all.

The only thing I know is that I know nothing.

Socrates

It would no doubt be on my part, dare I say it, a bit voyou, a bit roguish, if not roué, were I not to begin here by declaring, yet one more time, my gratitude.

Yet one more time, to be sure, but for me, yet one more time ever anew, in a way that is each time wholly new, yet one more time for a first time, one more time and once and for all the first time. Not once and for all, not one single time for all the others, but once and for all the first time.

Jacques Derrida
(Introduction to: *Rogues: Two Essays on Reason*)

Contents

List of Figures

List of Equations

Chapter 1

Introduction and Motivation

1.1 Articulating Knowledge in Groups

The genesis of this paper lies in thinking about thought and understanding – why are some descriptions and/or interpretations "reasonable" and not others? What underpins our modes of reasoned thought? How do a group of people develop aggregated theories and points of view, and specifically what mechanism(s) gives rise to such consensus? While this paper shall not dwell on the morass of philosophical questions that thusly arise, it begins its inquiry by examining one particular means for the articulation and storage of group understandings – that of ontologies. An ontology, in the context of knowledge engineering, seeks to express the understanding of a group of people using a shared vocabulary. Often, ontologies are implemented in some formal language, facilitating automated reasoning and machine readability.

A person's internal understanding of the world is expressed through some agreed upon language. Ideally, the agent's intuition is accurately captured in the language of the

ontology, and is thus accessible to others making use of it. When using a formal language to represent how we understand the world, we're often connecting objects in the world via relations. A rigorous and precise way to define the behaviour of these objects and relations is to then axiomatize them. This brings us to one of the overarching problems when using ontologies – the process of articulating one's internal understanding in the syntax, grammar – language – of the ontology is not always as straightforward as we would like. Most of what humans collectively know is stored in the minds of people who may not have had formal training in languages such as first order logic, or even Boolean algebras. Consequently, getting what one knows into the shared, common vocabulary governed by an expressive syntax and semantics is a difficult process.

The second overarching problem in ontologies is analogous to translating between spoken languages. When different groups of people use different syntaxes and vocabularies to express insights about the world, translating from one to another does not always preserve the contextual or idiosyncratic components of the original intention of meaning. Meaning with precision is one of the desirables of ontology use, and identifying what meanings are intended among multiple ontologies, developed by different groups, is important. Furthermore, this problem, referred to as generating semantic mappings, becomes particularly acute when one considers entities called ontology repositories.

An ontology repository serves as a library of ontologies, cataloging them and ideally their relations to one another via an overarching theme. Identifying these relations leads to a particular type of ontology, called Upper Level Ontologies. These representations strive to define fundamental, "upper" concepts that are then used by more specialized ontologies. As noted above, a formal language, such as first order logic is particularly well suited to capturing a lot (though not all), of the intuitive understandings of people describing how they realize the world.

1.2 Current State of Affairs

Looking at the types of ontologies developed today, and the way people use them, or rather hope to use them; the general trend in socio-technical systems is towards a system of data representation that emphasizes reusability via modularity and/or extensibility. Concurrent with these trends is the increased proliferation of more structured data formats and knowledge representation, often in the form of limited taxonomies, or "conceptual black boxes." Ideas and entities are named, but the meaning and behaviour of the name is largely accessible only outside the system of representation. The majority of these approaches do not capitalize on the advances offered by ever more powerful reasoners and logical paradigms. The reasons for this resistance are numerous, but one of the main factors that hinder the wide spread adoption of more expressive and precise descriptions of terms is a lack of widespread familiarity and facility with formal logic.

This brings us to the practical matters at hand, specifically what motivating problems exist in the ontology community today? We notice again that no matter how well someone understands a concept or phenomenon, the utility of their knowing is limited by their ability to express themselves. Our primary vehicle for the transmission of such knowledge is in the form of some language. Hence getting it out of our heads is the first problem; this corresponds to ontology generation. The second problem arises when we consider what happens next? How can we ensure that others understand what we're saying? This is the semantic mapping problem. This thesis hopes to alleviate many of the obstacles posed by the above two problems, using the idea of structural metaphors to drive the development of an ontology repository which will aid ontology designers in axiomatizing their intuitions and serves as a reference for which to identify semantic mappings.

1.3 Upper Level Ontologies and Ontology Repositories

Thus, we must take a closer look at how ontologies are generally manifested today – what support structures exist to aid ontology designers, and how are disparate ontologies

collated and/or compared? To advance the basic idea of reusability, numerous groups have developed or are developing Upper Level Ontologies (ULO), in an effort to establish frameworks where more domain specific ontologies may be derived (Herre et al 2006; Gangemi et al 2002; Niles & Pease 2001; Grenon et al 2003; Lenat & Guha 1990; Cui & O'Brien 2000; Pan et al 2003). While this approach is useful in many respects, it suffers from a series of shortcomings:

- there is rarely any agreement on the hierarchy of these "foundational" concepts
- they are often limited to defining an entity or object as a one word concept (not much is said about the nature / behaviour of the concept)
- it is often unclear how to compare disparate ULO's to one another

In response to these perceived shortcomings, this thesis capitalizes on the nature of metaphor and logic to develop a complementary solution; a quick word then on the role of metaphor.

1.4 Metaphor, Logic and Knowledge

It is the more expressively precise method of representation that is being explored in this work. For once explicit, First Order Logic (FOL) axioms are used to denote the behaviour of objects and the relations connecting them to one another, the idea of metaphor may be invoked and utilized to our benefit. While the question of what metaphors actually are is beyond the scope of this discussion, we draw one particular definition from (Lakoff & Johnson 1980; Danesi 2002; Pinker 2007) to ground our idea and define our objectives. This conception of metaphor is often called a "structural" or "conceptual" metaphor (Lakoff & Johnson 1980; Danesi 2002). In this context, the metaphor acts as a mapping of the underlying "reasoning journey" from one concept or framework to another.

Here the view is that our idea of concept A is an amalgam of many other concepts, tied together via a logical scaffolding, much like how buildings are built on a primary structure. One idea that this thesis advances is that the basic logical structures employed

in diverse fields actually (re)use the same form regardless of the particular domain. It is looking at logic as lego blocks, blocks which have yet to be properly identified, formalized or represented, but once done, would allow a much quicker, more precise, less ambiguous and ultimately rigorous exchange of assumptions, arguments and theories.

Metaphor (and analogy) is the most lucid example of this approach, where we map the logical substructure of one concept to another; while the names of the particular entities being compared differ. Often, it is the way in which the whole (or parts of the concept) are connected to one another is what is preserved through the use of metaphor.

1.5 Solutions?

Instead of creating an Upper Level Ontology based on a conceptual hierarchy that this or that group of authors think is appropriate, a bottom-up approach, based on logical modules is offered. The resulting support environment is actually more of a hybrid amalgam between ULO's and ontology repositories. In its simplest form, an ontology repository serves as a store for various ontologies, usually unified under a theme (such as BioPortal), ideally with logical relations defined between its member ontologies (BioPortal 2008). The proposed hybrid, ULO/repository creates an ontology for each "Core Hierarchy." Core hierarchies here refer to different theories represented in FOL or CLIF. For each layer of abstraction, there is a set of core hierarchies. Moreover, the limits of each hierarchy are demarcated by the fact that they share no non-conservative extension with another hierarchy at the same level of abstraction. Thus, each hierarchy would represent a series of modules which capture theories that extend a basic concept for that layer of abstraction.

For the purposes of this thesis, we will be developing the layer of abstraction for classes of logical structures that have been cataloged and defined by mathematicians, these structures include, but are not limited to:

- Orderings
- Groups

- Symmetries
- Geometries
- Fields
- Etc

Thus ontologies for each of these fields of mathematics would constitute a core-hierarchy. The ULO component corresponds to the support structure that allows users to draw out and reuse these "basic" logical constructs. It should be stressed that this is not proposed as a replacement for existing ULOs, but rather as a complementary environment to address some of their shortcomings. Moreover, one *could* also go to a lower level abstraction, specifically at the axiom level. For example, hierarchies at this lower level might correspond to "Binary Relations," "Ternary Relations," etc., with theories being "transitivity," "associativity," etc. While this is certainly a possibility, decomposing theories by axiom is beyond the scope of this thesis. It is left as future work, since the selected level of theories to implement (and those above), represents rich and well understood concepts that illustrate the ideas in the thesis clearly.

In effect, the concept of the ULO (and indeed metaphor) is being turned on its head. Instead of linking entities by their names, it is the logical substructure that will do the guiding. Moreover, by developing this ontology repository, the use of more expressive logical formalism (such as FOL), will hopefully be encouraged and achieve wider use, overcoming many of the problems already identified when people understand ontology as simply a "taxonomy" or "classification" scheme (Sowa 2007). Additionally, this mathematically precise and sound representation of the behaviour of entities allows us to apply discoveries in fields such as order theory or computer science directly, with a clear understanding of the models being described. In such a way, novel ontology designers, or those who wish to make their representations less ambiguous and more robust, may engage with this ontology repository to derive the most appropriate set of FOL axioms which describe the behavior of an entity or relation.

1.6 Assumptions

The basic assumptions of this thesis, the shortcomings of the current field of ontology and knowledge representation, as well as the functional objectives of the ontology repository are discussed in the following sections. The points below enumerate the basic premises guiding the work in this thesis:

- Ontologies represent explicit, shared understandings of the world
- Formal languages are well suited to express much of what we know
- Much of the world can be usefully represented as entities and the relations between entities
- First Order Logic allows us to represent much of the above in an unambiguous way
- Structural / Conceptual Metaphors map the logical substructure from one "concept" to another

It is acknowledged that these questions are to a degree still under investigation in philosophy, cognitive science, neuroscience, psychology etc, but for the purpose of creating something productive, they are being taken as points of direction. When cooperating in groups, and building knowledge bases accessible to many people, we shall rely on ontologies expressed in formal logics to capture and articulate what we think we know. Moreover, the idea of metaphor directs the organization and procedure of the axiom generation, by focusing on the logical structures that underpin disparate forms of explicated thought.

1.7 Limitations of Ontology Research at Present

Several issues and shortcomings that currently hinder research in the ontology community are discussed below. Firstly, we note that for many people, ontologies are currently stuck in "taxonomy" or "classification" mode. This means that in numerous cases, the only available relation is *is_a*, (and perhaps a few other predefined relations).

Such limitations seriously hinder the ability of an ontology to satisfactorily represent a domain of knowledge.

Indeed, while languages such a Resource Description Framework (RDF) and the Web Ontology Language (OWL) are relatively easy to use, they further exacerbate the problem above. While they have the advantage of being decidable, they often lack sufficient expressivity to reflect the world. Consequently, much of the intended semantics of many concepts cannot be represented in an ontology. Storing these semantics external to the system of representation can introduce significant hurdles when trying to apply more involved reasoning or even reusing an ontology among different groups of people.

One direct consequence of this choice in language is that the problem of Semantic Mapping becomes far more complicated. People are often relegated to defining vague notions of similarity to compare two concepts in different ontologies because so much about them is left unsaid. Similarly, the potentials of ontologies seems to be curtailed by such limited representation systems.

At the same time, formulating an ontology using a more expressive language such as FOL can be a difficult process. Whereas RDF and OWL have relatively clear guidelines as to what constitutes an axiom and how they ought to be written, FO theories offer no such similar support. Moreover, a paucity of training regarding formal logics also presents an additional barrier to people using these languages to express their intuitions.

Finally, in trying to address the problem of reusability and extensibility, research on Upper Level Ontologies has thus far largely focused on top-down reusability. These approaches have led to an oversight in identifying many reusable components of logic based knowledge representation. While some research in semantic mapping has considered the bottom up approach (Stumme & Maedche 2001), none appear to have considered using these as bases of ontology building. Metaphor and logical structures built bottom-up address this oversight. A significant portion of this thesis is about identifying those theories that are of interest for they are re-used again and again in these

logics. These blocks may be viewed as the natural developments or biases of the language being used, in this case for FOL.

1.8 Functional Objectives

The goals and functional objectives of this thesis are also listed below:

- Develop an extensible Ontology Repository / ULO that provides people with basic modules of logic
    - Create an ontology repository whose organization reflects the classification of models of particular first-order theories
        - Use this repository to propose/determine the strongest set of axioms that capture a user's intended semantics for a particular relation or function and are extensions of the repository's core theories
        - Use this ontology repository to propose/determine semantic mappings between ontologies that are extensions of the repositories core theories
- Allow ontology designers who are unfamiliar with FOL to develop complex FOL formulae via an easy to use, intuitive interface
    - Users need not be familiar with the syntax or grammar of FOL, nor need they know the theories in the repository, they only need to be able to construct at least one model of what they are trying to define as well as being able to discriminate whether proposed models are acceptable.
- Allow the collection of novel axioms to complement existing first-order theories
- Encourage and facilitate the increased use of FOL in knowledge representation
- Educate ontology designers in *one* perspective of how thoughts are employed and used

The implementation of these objectives as a result of this thesis is limited to a proof of concept for the ideas developed. Consequently, in its first iteration the repository will be

limited to one ontology, namely a subset of Order Theory pertaining to Partially Ordered sets (posets). An algorithm has been developed that allows the repository to learn, via interaction with an agent, their desired semantics. This algorithm applies in general to any logical relation that can have models manifested in some unambiguous, accessible form. Posets correspond to binary relations, and thus have models easily manifested in visual form, via graphs (or perhaps Hasse Diagrams). Consequently, a simple program has been developed that allows ontology designers to generate *models* which are then translated into FOL statements. The current implementation uses graphs to represent models, and is thus better suited to capture binary and some ternary relations; however it should be noted that in general *any* n-ary relation that may have models representable in some accessible form is under the purview of this algorithm. In addition, another algorithm has been developed to facilitate semantic mapping between ontologies using the repository as a mediator.

Finally, to facilitate an open and extensible repository, guidelines have been developed to formalize the process of maintaining and growing such a structure. It should also be noted that significant user testing and feedback is not available, so from a design perspective, this system is being largely created *a priori*. Hopefully, the ideas underpinning the current implementation will attain critical mass, and contribute positively to the greater ontology and scientific community.

Chapter 2

Background and Literature Review

This chapter will provide the reader with the necessary background to follow the scope, successes, limitations and goals of this thesis. We will briefly revisit what an ontology is, before considering the first problem identified by this thesis; that is, explicating the different elements involved in articulating and writing an ontology. The question of how one might define and determine a semantic mapping between two or more ontologies is also explored. For each of these components, the current state of research in each field (to the detail relevant for this paper) will be provided, as well as a brief discussion of the limitations and biases in each. They are collected in the Proposed Solutions section. The problems and previous solutions are evaluated and summarized in the final section. The choices made for the ontology repository developed in this thesis shall be discussed in greater length in the ensuing chapters.

Before we begin, we note that this thesis is situated in the context of ontology research and practice today. Thus we must consider the different ways they are organized, and what forms ontologies have thus far taken. How might one articulate her/his intuitions in

an unfamiliar grammar and syntax? Expressing ideas would seem to be quite difficult. Assuming something has been expressed, how might we then define and possibly generate automated semantic mappings?

Among the solutions explored, researchers have proposed various language families to address the problem of articulation and templates to guide the axiom generation process. Furthermore, Upper Level Ontologies and Ontology Repositories have served as vehicles to facilitate ontology reuse. Several tools have also been developed by other researchers, coining terms such as ontology mapping, alignment, integration and merge to address and better understand the problem of semantic mappings.

In all, this thesis seeks to solve some of the problems in ontology design and semantic mapping by leveraging the idea of metaphor to construct a novel ontology formulation method. To that end, it is important to revisit issues in metaphors, the grounding of formal logics and previous attempts at interfacing and visualizing logical formulations.

2.1 Ontologies

An oft-cited description of ontologies is provided by Gruber, where he states that ontologies are "an explicit specification of a conceptualization" (Gruber 1993). Here, the terms conceptualization and specification are left somewhat ambiguous, though Gruber does suggest that the universe of discourse and its corresponding vocabulary are drawn from a domain of knowledge and represented in a "declarative formalism." The following definition given by Uschold and Grüninger gives light to a more thorough understanding of what ontologies may afford:

> *'Ontology'* is the term used to refer to the shared understanding of some domain of interest which may be used as a unifying framework to solve the above problems.... necessarily it will include a vocabulary of terms and some specification of their meaning (i.e. definitions) (Uschold & Grüninger 1996, pp 5-6)

The question of what is knowledge, and the nature of the relationship between "knowledge" and its' users is left for philosophical investigation. This interpretation of ontology allows engineers and designers to appropriate some basic concepts to create useful tools to capture the essence of these questions. That the knowledge of many domains may consequently be represented in a fairly formal manner, using a well specified lexicon and relationships gives rise to an area of research and analysis that helps both reproduce said knowledge more easily in the future, and to better understand the knowledge domain.

In this context, ontologies, the systemic account of the domain knowledge strives to comprehensively capture all objects (any entity) and relations that govern or describe the interaction between objects (Genesereth & Nilsson 1987). Such a representation of a system would ostensibly deliver the following:

- A thorough analysis of the domain knowledge (limited by the expressiveness and precision of the language)
- Distinguish domain knowledge from operational knowledge
- Make domain assumptions explicit
- Enable reuse of domain knowledge
- Create a shared, common understanding of the structure of information among either people or software (Musen 1992)

The implementation of these goals often takes the force of representing objects in the world and identifying the relationships between them. Not surprisingly, the basic blocks of most ontology languages are "classes" (or "objects") and "relations." Classes are generally representations of objects found in the world, one would be tempted to use the word "concepts," but there is a philosophical quibble with this semantic choice. They consist of a set of objects and can have either classes or individuals are their members. An example of a class might be Car.

It is here that some approaches to ontology design diverge, largely due to the choices in the language used. In the context of computer and information sciences, with a strong emphasis on machine readable knowledge representation, formal languages are desired. Choosing which of the various available representation media to use in writing an ontology leads to different biases, and generally represents a trade off between decidability, tractability and expressivity. The issues concerning writing and articulating ontologies are discussed in the following section.

2.2 Articulating and Writing Ontologies

When one wishes to develop an ontology, the first choice is in picking the language in which to communicate. As noted above, formal languages are those used in ontology representation as natural languages are still too ambiguous and dense for many types of machine reasoning. The advantage in using formal languages is that the semantics are ostensibly clear, with little room for confusion. On the other hand, this precision limits the types of concepts that may be easily communicated in such a medium.

Moreover, since we are interested in reasoning, some sort of logical system is an attractive option. While logic in some form or other can trace its lineage some 2500 years, the formal logics employed in this work were only first articulated over the past 100 years. Today, a multitude of logics have been developed, each with their own unique advantages and short comings. In the section on ontology languages below, first order logic (itself only 70 or so years old) will be compared to its pared down sister of description logics.

Within formal logics, those that have working automated theorem provers (ATP) are prioritized, since they allow automated reasoning. Most of these languages fall along a continuum, representing a trade off between decidability and expressivity. The discussion of this thesis is limited to FOL and less expressive logics, since among other issues; the dearth of ATPs for modal and higher order logics reduces their appeal for ontology representation – for the moment. The rest of this section will discuss the difference

among common ontology languages, as well as providing greater information regarding the syntax and semantics of the language used in the ontology repository, CLIF.

2.2.1 Ontology Languages

At present, there is a dichotomy between ontologies based on a more straightforward implementation of FOL (i.e. CLIF) - which generally differentiate between "relations", "functions" and "objects" - and other ontology languages (i.e. OWL, RDFS) that instead use terms such as "classes", "instances", "attributes" and "relations." This discrepancy between first order logics and description logics leads to two loosely dissimilar understandings of what ontologies allow.

A widely used family of formal languages is description logics, a term coined in the early 1980's. The first description logics were developed in the 1980's by (Brachman & Schmolze 1985), and have gradually become more expressive, including more first order expressible concepts (Gangemi et al 1993; Nardi & Brachman 2003; Baader et al 2003). They represent formalisms that are less expressive than traditional propositional logic and predicate calculus, yet work well in categorizing and classifying "is_a" or subsumption relations between objects. Additionally, most DLs have Boolean operators, and some form of quantification. Description logics were developed in response to the absence of formalism in many semantic networks. While they are closely related to propositional and modal logics, and they only employ fragments of first order logic (FOL), having grown out of the optimization of tableau algorithms, where decidability was often a large concern (Baader et al 2003). Examples where DLs have been implemented include KL-ONE, OWL, etc. (Brachman & Schmolze 1985; Patel-Schneider et al 2004).

The latter ontologies, based on description logics, generally capture the notion of "is-a" or "contained in" relations; although what the idea of "is-a" actually means isn't always clear. Ontologies derived from these carry a taxonomy bias, where they categorize objects in hierarchies. Lacking the greater expressivity afforded by other logical formalisms, relations may simply boil down to naming something, missing important

restrictions on the behaviour of that object. When the logical vocabulary is limited to only “is-a”, “contained-in”, and some snippets of FOL (i.e. “transitive-property”), the picture of the world that emerges is sometimes either too simple or ambiguous. Ideas being represented become a form of “conceptual black boxes,” with a significant amount of information left external to its ontological expression.

Ontologies based on axiom generation in FOL allow for greater precision in describing the properties of objects and relations. They enable the formulation of more complex logical constructs, such as partial orderings, comparability graphs etc. Although they too are restricted in expressivity in comparison to higher order logics, they offer much more nuance than comparatively restrictive languages. Three common ontology language families are briefly evaluated next. It should be noted that this thesis chose to represent its “knowledge” using the Common Logic Interchange Format (CLIF), a relatively direct implementation of FOL (CL Standard 2007).

2.2.2 Extensible Markup Language Schema (XDS)

One such idea which is not a language in itself but a framework in which to define semantics is based on the Extensible Markup Language (XML). XDS allows users to construct machine readable documents in a structured way (Farrell & Lausen 2007), being intended to enable users to create their own custom markup languages. Unfortunately, it also imposes no semantic constraints on the documents it is describing. While XDS has introduced a series of data types and has restricted the structure of XML documents, it still does not afford adequate expressivity to represent many intuitions (van der Vlist 2001).

2.2.3 Resource Description Framework Schema (RDFS)

RDFS was developed as an extension of RDF, which itself was initially designed to be a metadata data model, used for describing web resources (Hayes 2004). With the creation of RDFS, it has been increasingly incorporated as a general knowledge description language. The basic conceptual building blocks are structured using triples – in a Subject-

Predicate-Object form. Here, the predicate is similar to a relation, describing how the subject relates to the object. Thus the statement "the sky is blue," may be mapped into RDF where sky = subject, is = predicate, and blue = object.

Some projects such as Dublin Core (DC) – which is used to denote authorship information about journals, books, articles etc – use RDFS to generate their ontologies. While very effective in this capacity, the types of statements made are generally restricted to A "is-a" or "has-a" B. The nature of the predicate is often not represented explicitly within the ontology. This allows RDF ontologies to be easily written in XML, greatly enhancing its accessibility to the wider web community; however the semantics it permits is generally limited to generalization hierarchies of predicates and classes.

2.2.4 Web Ontology Language (OWL)

According to the official World Wide Web Consortium (W3C) document on OWL, it was designed "for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics." (McGuinness & van Harmelen 2004)

OWL seeks to address some of the shortcomings it identified in its predecessors (as above). Consequently, it has a greater vocabulary to describe classes and relations, notably, including ideas such as "disjointness," "cardinality," "transitive-property" etc. OWL in many ways straddles the line between RDFS and first order logic. As stated earlier, OWL DL and/or FULL, the most expressive manifestations of the language family, simply define vocabulary terms such as "transitive-property." Instead of allowing users to describe a concept like "transitive-property" themselves, it is sort of "given from above." As a result, it is exceedingly difficult to define novel properties of the same (let alone higher) complexity in OWL.

2.2.5 Common Logic (CL)

The last of the dominant ontology languages introduced here is Common Logic (CL), which defines a standard semantics for its dialects. Of these dialects, the syntax implemented in this thesis is that of Common Logic Interchange Format (CLIF)). While Common Logic shares many similarities with Traditional FOL, it has a few subtle tweaks that differentiate it as well. In contrast to most first order languages, CL has

> *a syntax which is signature-free and permits 'higher-order' constructions such as quantification over classes or relations while preserving a first-order model theory, and a semantics which allows theories to describe intensional entities such as classes or properties.* (CL Standard 2007)

This in turn is a result of a key distinction between the symbols used in representation, and what they refer to in the meta-theory. Whereas in FOL, relations define sets, CL treats relations as names referring to objects in the universe of referents (UR), which is a superset of the universe of discourse (UD). This distinction means that when quantifying over relations, one isn't in the awkward position of trying to include a set as a member of itself.

While the above affords greater expressivity in CL, it should be noted that any FOL theory can be represented equivalently in CL. However, the greater expressivity of CL in the formulation of a theorem means there is no guarantee that an equivalent one is available in FOL. Common Logic differentiates these cases by referring to dialects which use non-discourse names as *segregated dialects* (corresponding to TFOL). Similarly, Description Logics are easily expressible in CL but not the reverse. Furthermore, any structure articulated using XDS, RDFS or OWL can be translated into CLIF, although the same cannot be said in reverse.

The syntax and semantics of Common Logic (and CLIF) are detailed in the following two subsections; the reader is directed to (CL Standard 2007) for a more thorough treatment of CL.

Language	Expressiveness	Decidability	Prevalence	Relation to Formal Logics
XDS	User Defined	User Defined	Medium	User Defined
RDF	Low	Decidable	High	Various Bits of Formal Logics
OWL	Medium	Decidable	Medium	Fragments of FOL
CL	Relatively High	Semidecidable	Low	FOL+

Figure 2-1 – Language Comparison

2.2.5.1 Syntax

The CL framework does assert some conventions, such as using quotes to denote strings and interpreting XML tagged data. However, the CL family of languages does not specify a proof theory or inference rules.

At its core, CL (and thus CLIF) consists of a series of logical and non-logical symbols. The logical symbols are themselves composed of logical operators, quantifiers, parentheses, an infinite set of variables and an equality symbol. The logical operators in CLIF include {and}, {or}, {not}, {if} → and {iff} ↔, while the quantifiers are {for all} and {exists}. All operators (except for equality) are also names and terms in CLIF.

Non-logical symbols consist of constants, predicates (relations) and functions. In CLIF, these operators fall under the scope of *interpretable names*. Constants map these abstract symbols in CLIF to ones such as “man” or “0” etc. A predicate expresses a relation between these variables, returning a True or False depending on whether the relation holds or not. Each predicate is said to have an *arity* or valence of value n, where n corresponds to the number of variables taken in its argument. Finally, a function is very similar to constants, however whereas constants have a valence of 0, a function has a

valence ≥ 1. These three different methods of pointing to things in the world may lead to various ways of expressing the same idea. For example, many subclasses of partially ordered sets use the idea of a "meet" and "join", which are often defined as functions. Alternatively, other order theories do not use these concepts and instead use relations such as minimal upper bound and maximal lower bound.

The statements below show the relationship between meet-join and mub-mlb, an equivalence (in some cases), which is exploited in this thesis:

(forall (?x ?y ?z) (iff (mlb ?z ?x ?y) (= (meet ?x ?y) ?z)))
(forall (?x ?y ?z) (iff (mub ?z ?x ?y) (= (join?x ?y) ?z)))

Equation 2-1

In the first statement above, z is a variable, which is equivalent to the result of the function which evaluates the meet of x and y. The relation mlb, notes that z is the maximal lower bound for x and y. All functions may be represented as relations, but not the converse. A function necessarily has a unique element that it points to; relations make explicit the thing a function points to, and does not necessarily impose uniqueness.

In traditional FOL, two constructions are usually defined: *terms* and *formulae*. In CLIF the notion of formulae become squished in to the definition of sentences. Terms are defined as follows:

1. any constant is a term (and an argument)
2. any variable is a term (and an argument)
3. any expression $f(t_1, t_2, \ldots, t_n)$ with $n \geq 1$ arguments, where t_i's are terms, and f is a function symbol of arity n, is also a term
4. nothing else is a term

Instead of using formulae to define what snippets of logic are acceptable, CLIF uses the notion of atoms and builds on them. Atoms and operators are then used to define acceptable sentences. In general, a sentence is defined to be a unit of logical text that is either *true* or *false* in an interpretation. A sentence can further be divided into one of atomic sentences, Boolean sentences, quantified sentences or a comment sentence. An atomic sentence is either an equation or an atom and is represented by role-pairs consisting of a role-name and a term. Boolean sentences use operators, and represent disjunctions or conjunctions of terms. Quantified sentences are those that have a truth value and make use of quantifiers. Finally, comment sentences are ones where a string of text is inserted in quotations.

Common Logic also defines a hierarchy of string types, from the most general *text* (a set, list, or bag of phrases), then to *phrases*, which consists of *modules, importations, sentences* and *comments*. The first three are different ways of referring to a body of logical text, while the last identifies text that allows the expression of ideas external to the language of representation. For a thorough treatment of the particulars of CL and CLIF, the reader is encouraged to consult the ISO24707 document (CL Standard 2007).

Now that the basic syntax of CLIF has been outlined, we must turn attention to how meaning is assigned or derived from these symbols. That is, we must consider the semantics of this language, which is elaborated in the following section.

2.2.5.2 Semantics

The above are the essential constituents for the construction of theories using CLIF in CL. However an additional conceptual framework is required to translate the above syntax into meaningful representations of the world. It is useful to take a step back and consider what logical representations attempt to achieve.

When articulating an idea using a formal language, we develop a system of symbols to refer to objects. The relations and functions governing how these objects interact with

one another are reflected by the logical operators we use to connect them. In TFOL, a universe of discourse (*UD*) is used to refer to objects in the universe. An interpretation, *I,* maps a set of sentences *Φ* to elements in *UD* leaving certain non-logical symbols, i.e. relations outside the *UD*. In contrast, CL employs an ingenious trick by defining two universes; that of discourse (*UD)* and referent (*UR*) and using a function, rel^I to map names in *UR* to entities in *UD*. What this separation means is that we explicitly make clear that the symbols that we are using are not the same as the objects we are considering. In segregated dialects however (such as TFOL), *UR* and *UD* are not the same, since relations are not in the *UD.* With this distinction in mind, if we use rel^I to quantify over relations, the paradox of a set being in itself is rendered meaningless.

The following example should illustrate the difference concretely. Let us consider a non-segregated dialect where:

$$UD = UR = \{1, 2, fred, human, unary, lessthan, transitive\}$$

in this case,

$$rel^I(human) = \{fred\}$$
$$rel^I(unary) = \{unary, human, transitive\}$$
$$rel^I(lessthan) = \{1,2\}$$
$$rel^I(transitive) = \{lessthan\}$$

Equation 2-2

Note that while "unary" is in the extension of "unary," the extension of "unary" is not the same as the name (relation) "unary". In contrast, the *UD* in a TFOL construction would not include relations, and would only be {1, 2, fred} since if "unary" were included, we would be in the awkward position of:

$$unary^M = \{human, unary, transitive\}$$

Equation 2-3

which is a set containing itself. Thus we could conceivably, continually replace "unary" with {human, unary, transitive} *ad nauseum*. Thus, TFOL forces certain non-logical symbols (i.e. "human") to be outside the universe of discourse. In such a segregated dialect, *UD* is contained in *UR*, and there will be some names which are not in the former (i.e. relations).

In both types of dialects, an interpretation assigns a denotation (extension of a set) to some (or all) of the non-logical symbol in the universe of referent. This denotation is achieved by referencing a non empty universe of discourse *UD*, where each constant is mapped by a function to an element of *UD*, each n-ary operation or function to an n-ary operation on *UD* and each sentential symbol to either True or False.

An additional idea of a *model* is also useful for talking about the semantics of a theory. In logic, a model *M*, consists of a pairing *<UD, I>*, where as above, *UD* is the set of elements populating the universe of discourse, and *I* is an interpretation which maps sentences in the language to elements on *UD*. If a model can be constructed for a theory, it is said to be satisfiable. If every possible model and interpretation of its variables is true, then the theory is also said to be valid. This thesis will be drawing heavily on this component of languages. The relationship between models and their corresponding axioms points a way to communicating ideas in FOL without having to explicitly know the language.

It is important to also note however that one other ingredient in establishing a formal language is the need for a series of inference rules which allows one to prove theorems. These rules act as a function which maps sets of formulae to sets of conclusions. They are external to CLIF and Common Logic, and are instead defined and contained in theorem provers. The mechanics of the algorithms and rules decided by provers are outside the scope of this thesis and comprise part of a larger effort to insure that the scalability and tractability of this approach is manageable.

Writing axioms using CLIF or any other FOL implementation usually demands that a user be well versed in the syntax that defines the language. The semantics of an axiom or a set of axioms are expressed in the interpretation. One of the main drivers of this thesis is to bridge the chasm that exists among casual users of logic, between its syntax and semantics. While many people are able to express or articulate the meaning of their intuitions in natural languages, via pictures or some other means, the syntax of FOL often presents an impediment. While some have worked towards visualizing the *syntax* of FOL, we focus on manifesting the *semantics* of FOL in accessible ways to ontology designers.

In order to address this acute concern, the ontology repository discussed below aims to present the *semantics* of the underlying logical patterns to users, instead of forcing them to navigate potentially unwieldy syntax. This is achieved by leading the user through a series of yes or no questions based on models generated by classes of axioms. The details of this approach are outlined in the discussion regarding the ontology repository and design algorithm (chapters 3 and 4).

2.2.6 Choosing the Right Language

Figure 2-1 above illustrates the similarities and differences between the four ontology language families discussed. In choosing which language to use for this representation, it may also be useful to consider some criteria of good ontologies that Gruber has specified (Gruber 1993):

- Clarity
- Coherence
- Extendibility
- Minimal Encoding Bias
- Minimal Ontological Commitment

The languages are arguably, relatively equal for the first three criteria; however, when considering Minimal Encoding Bias, which Gruber defines as:

> *The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. An* encoding bias *results when a representation choices* [sic] *are made purely for the convenience of notation or implementation.* (Gruber 1993)

All the above languages impose some degree of encoding bias, due to their explicated inability to thoroughly document what we would like to represent (i.e quantification over propositions). However, the bias in the first three language families is even more pronounced due to their limited expressivity. Consequently, one finds that many ontologies developed using RDFS or OWL, often fall into a taxonomic / classification trap.

To better realize and maximize the potentials that ontologies offer in assisting humans, whether within the sciences or in business, to communicate across national, cultural and technical boundaries, the increased use of more precise and expressive ontology languages would be seemingly desired. Considering that the relation "is-a" is oftentimes ambiguous, and can depending on who is using it, can take multiple different meanings; the shortcomings of the first three language families becomes readily apparent.

Within the context of developing an ontology repository that allows people to identify and formulate axioms that hinge on basic logical building blocks, one particularly suitable language family is Common Logic. To reiterate, the idea that this thesis seeks to exploit is that to a large degree, humans (especially within the sciences or other consensus and reason based disciplines) understand one another logically. Admittedly, this cannot be a thorough treatment of how we understand, as there are some things that can't be articulated in first order logic; however the set of things that we *do* understand and communicate with one another that can be described using FOL or pseudo-SOL, is enormous and has been barely tapped. While increased numbers of people realize the potential of a semantically enabled machine world, too many are naïve about their choice of language and how it biases and limits what they would like to express. Although the

idea of language biasing what can possibly said is less plausible for natural languages (Sapir 1929; Whorf 1940), when one considers formal languages which impose strict restrictions on expressivity, this idea gains greater currency. First order logic will never be able to express an idea that requires quantification over propositions; for if it were to do so, it would no longer be FOL.

2.2.7 Difficulties in Articulating Ontologies

One impediment that gives rise to this linguistic naivety is the perceived difficulty in formulating FOL consistent axioms. As noted by Hou et al, "research in classifying and representing axioms in a user-friendly way has been relatively sparse in the knowledge base-systems community" (Hou et al 2005). Two notable attempts have been made in this regard, both attempting to develop patterns and templates for axiom formulation based on existing ontologies. According to Hou, Staab and Mäedche, many of the axioms they encountered shared similar patterns, which they then abstracted to classes of axioms. These classes were then predicates that users could reuse (Staab & Mäedche 2000; Hou et al 2005).

One such example is EZPAL, an "environment for composing constraint axioms by instantiating templates" (Hou et al 2005). Having analyzed a number of ontologies that utilized FOL axioms, their group developed a template format for Protégé, where users could "fill in the blanks" to specify the axioms they wanted. Essentially, they categorized the axioms and translated them into English, using abstracted names, while also providing concrete examples of the axioms in use. Users would then only need to identify which template best corresponded to their intuition, fill in the blanks, and their requisite axioms would be generated.

This paper takes three subtle but importantly different approaches. The first is in the way the categories of patterns have been identified and selected for representation. This thesis seeks to utilize "patterns" already defined and well catalogued in mathematics and computer science. While it is not limited to these, building on these foundations allows

the interactions between the different classes of patterns to be better understood and evaluated in a formal, rigorous manner.

The second difference lies in the user interface being proposed for its implementation. While English language representation of what each class of pattern imposes will be provided, an effort will be made to visualize, if not the entire pattern, then the individual axioms that comprise the logical pattern. As the authors of EZPAL themselves note, "some participants suggested that the properties are too abstract, but those who disagreed said that picturing the relationships among frames mentally or actually drawing pictures helped in choosing the properties" (Hou et al 2005). It is believed that by using this type of pattern representation (through models), communication of FOL intuitions will be facilitated, allowing ontology designers to develop suitable axioms more easily.

A further refinement is that the interaction with this environment is not necessarily template driven. It is conceivable that an ontology designer may not have formulated the restrictions or behaviour of their concept in even pseudo-first order logic language. By providing an additional environment where users may "draw" the behaviour of their concept *in situ*, and then translating that drawing to the most appropriate class of axioms, a larger number of users should be able to incorporate FOL axioms into their ontologies.

2.3 Semantic Mappings

2.3.1 How can ontologies interact?

Immediately following with the first problem of articulating one's intuitions, is interpreting what others have said. Different people often interpret and represent the same phenomenon in diverse ways. While this is obvious when considering representations across domains, even within one, alternative characterizations may abound. Semantic mappings are attempts to explicate the relation between vocabularies and ontologies developed by different groups of people. In general, three types of ontology interaction have been identified (Pinto et al 1999):

- *Integration* – building a new ontology by reusing parts of others
- *Merge* – unifying multiple ontologies about the same subject
- *Use* – developing an application that can navigate different ontologies

Allowing multiple ontologies to work seamlessly with one another necessitates the creation of a mechanism to allow for the translation or mapping from one to another. The large number of ontology representations also encourages the development of automated mapping as the scale becomes unwieldy for manual translations.

Looking at the same problem from a slightly different perspective, Choi et al identify the following three types of ontology reuse:

- ontology merging – create a single coherent ontology from two or more existing ontologies related to the same subject
- alignment – creating links between two original ontologies while maintaining consistency
- integration – generate a single ontology in one subject from two or more ontologies in different subjects (Choi et al 2006)

From these, they identify three different types of mappings as well:

- mapping between an integrated global ontology and local ontologies
- mapping between local ontologies
- mapping on ontology alignment and merging (Choi et al 2006)

The first case by their account is to link views in the local ontologies to a query in the global ontology. In contrast, mapping between local ontologies involves defining a semantic relation between concepts in the target and source at a conceptual level. The final type of mapping entails determining the set of overlapping concepts, synonyms or unique sources among the various target ontologies that correspond to one another. In principle a ULO is supposed to serve this purpose of mediation from a universal basis to

particular domain specific theories. However no common standard has yet been developed, and as noted earlier there is little consensus on which ULO concept hierarchy is most appropriate or accurate and whether it even makes sense to work within only one ULO.

In an attempt to provide quantitative measures (or approximations) of a sense of correspondence, some researchers also invoke a concept of concept similarity and distance (Wache et al 2001; Stumme & Maedche 2001; Kalfoglou & Schorlemmer 2003; Kalfoglou & Schorlemmer 2003). Most research in this field seems to be concentrated on generating semantic mappings for ontologies developed using a variety of description logics (Pinto et al 1999; Wache et al 2001; Stumme & Maedche 2001; Madhavan et al 2002; Kalfoglou & Schorlemmer 2003; Noy, 2004; Noy & Stuckenschmidt 2005; Shvaiko 2005; Choi et al 2006; Giunchiglia et al 2007). It is then unsurprising that one common problem encountered by automated mappings arises when the language used to express an ontology is of limited expressivity, and the semantics are unclear. This ambiguity results in significant knowledge being contained external to the system of representation.

2.3.2 Sense of Integration and Mappings Used

The sense in which Semantic Mapping is invoked through this thesis is that of *use* as defined by Pinto, and that of *alignment* in the language of Choi et al. That is to say, the algorithm developed does not generate a novel ontology about a particular domain. It only suggests how two ontologies may interact with one another. It creates a set of links that maintain or highlight the parts of two or more target ontologies that are consistent with one another.

These links are procured by combining two of the senses of mapping as identified by Choi above. While the algorithm does not use an integrated global ontology, it uses a referent ontology to first align ontologies to a commonly understood set of conceptual theories. It then provides the user with an alignment of the target ontologies as

understood (if applicable) by the ontologies in the repository. The mechanics of this process are described the semantic mapping chapter.

2.4 Proposed Solutions

We begin by consider tools that have been developed to facilitate ontology and axiom generation. Attention is then shifted to two meta-ontology support mechanisms, which assist in ontology reuse and collation, notably, upper level ontologies and ontology repositories. Next, we note the role metaphors can play in mapping logical substructures of one concept to another.

2.4.1 Interfacing and Visualization

2.4.1.1 History and Larger Context

One of the most important components of written language is that it allows people to abstract the external world to a set of manipulable symbols. The tools that mediate the interaction between these symbols and our thoughts play a significant role in the adoption and proliferation of languages (Dewey 1938; McLuhan 1964). In the shift from oral cultures to those based more on the written word, three different approaches arose: phonetic, syllabic or hieroglyphic. In contrast to natural languages, formal ones such as FOL have been developed not with the oral tradition in mind, leading to manifestations that are superficially inaccessible to many.

In each case, until the 16th century, the relationship between the physical form and the language was mediated by quill and hand. The development of the printing press and then the computer fundamentally altered the relationship between the creator and the form of the language. Within the scope of this endeavor, this means that we now have at our disposal novel ways of storing, writing and communicating meaningful ideas. Software may now relatively easily and quickly generate models, or it might direct users in real-time as they are writing their axioms. In trying to leverage the opportunities afforded by

our dominant recording media today – the computer, its constituent software and peripherals (i.e. monitor, keyboard etc.) – we are inclined to ask the following:

1. what form is best for the presentation of models to the user
2. how should they be arranged (visually)
3. how might one develop an interaction environment that reduces the intention-language-meaning translation burden

None of these three questions shall be answered comprehensively, but they will be used to guide the functional principles of the developed environment. We note that the sense ratios that any presentation invokes in a user will, likely generate a consequential bias in the derivation of meaning from a picture or symbol. Keeping the above in mind, graphs consisting of vertices and edges seem like natural first candidates for the expression of abstract relations between two objects.

2.4.1.2 Templates

Several researchers have noticed the disconnect between what people want to represent, and what they are able to represent with adequate facility (Hou et al 2005; Sowa 2007). The two areas of concern are then, how do people input their intuitions, and how will they verify / visualize their inputs.

In trying to encourage people to utilize more expressive axiomatizations for their intuitions, Hou et al developed a template based system called EZPAL. It is interesting to observe that they noticed that:

> *many ontologies simply lack user-generated axioms, likely because axioms are very hard for developers to write. For example, in Ontolingua, only 20% of the ontologies provide manually generated axioms. Second, most*

> *of the axioms that do exist in the ontologies can be described by a small set of templates.* (Hou et al 2005)

By scouring the few existing ontologies that had used FOL axioms, they identified 20 manifestations that were repeated, and based on these, developed 20 templates. Users would then interact with the EZPAL software environment and generate axioms derived from one of those templates by "filling in the blanks." A screenshot of this interface is provided in Figure 2-2 below.

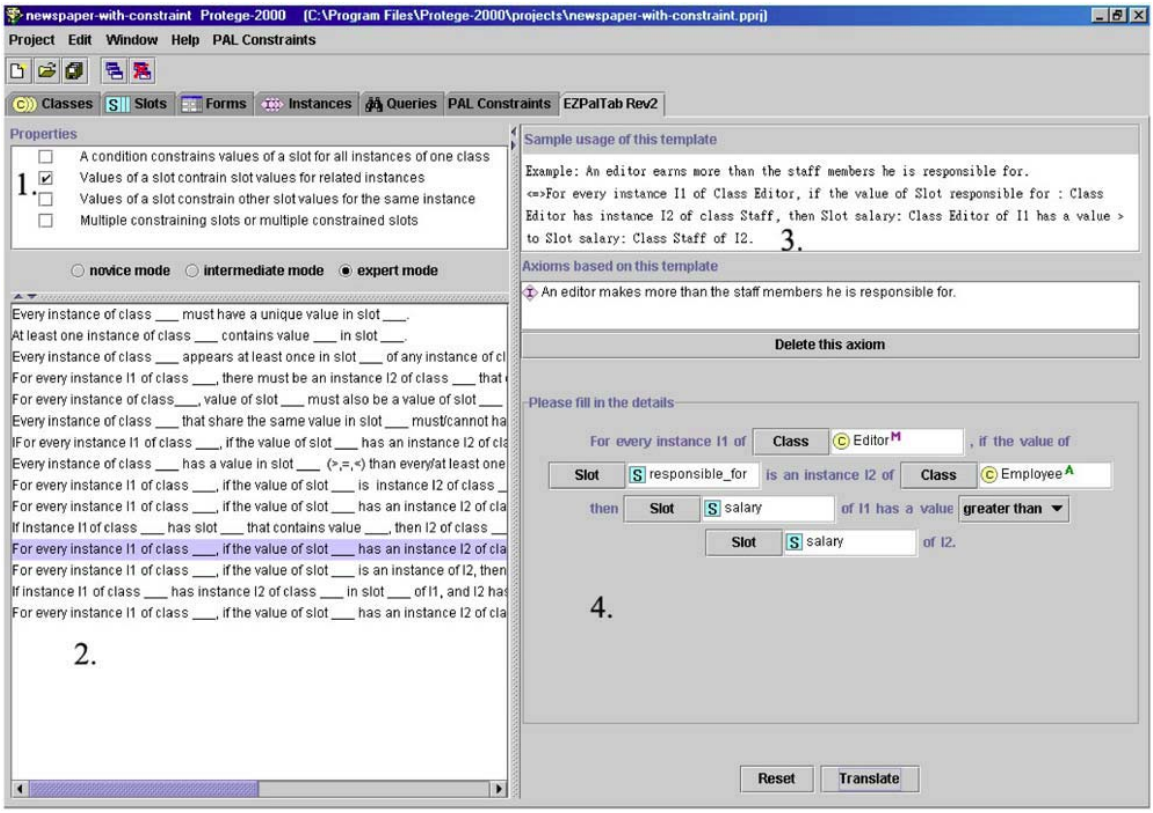


Figure 2-2 – EZPAL Screen Shot

A library of templates is available for browsing in panel 2, with an example of the meaning of the axioms shown in panel 3. The user would then fill the details of their intuition in panel 4. The main difficulty is in getting users to recognize the appropriate template. Several ideas were discussed, including developing a hierarchical metastructure to organize the templates; however, this idea was dismissed because the interactions

between the templates were not particularly clear. Instead, users were given English descriptions of the axioms, an example is provided below:

> *''Value(s) of a slot constrain other slot values for the same instances.'' This property applies when a slot value of one instance restricts the slot value of another slot for the same instance. For example, for every instance, slot ___ and slot ___ cannot contain the same value.* (Hou et al 2005)

They tested their environment by having six non-expert users try to encode 9 axioms provided in English, with 2 of those not being encodable through EZPAL. In short, this type of method provided significant benefits and allowed designer to relatively quickly and accurately derive the appropriate axioms (Hou et al 2005). Similar research has been conducted by Staab and Maedche in identifying patterns in FOL axioms, and developing predicates (such as PartonomicRolePropagation) that encapsulate those axioms, thus removing the need for designers to use the symbols in the axioms (Staab & Maedche 2000). Others working with the same motivation include (Barker et al 2000; Clark et al 2000; Staab et al 2000).

Most of these approaches are based on identifying knowledge patterns and allowing the designer to work with those patterns instead of the particular components of the pattern. The similarity between this abstracted representation of a complex idea and the idea of a *metaform* are striking (Danesi 2002). For some reason however, all the above approaches would rather that the ontology designer import not the underlying axioms that conceptualize a concept, but only the *metaform* into the target ontology. This is a curious choice as it externalizes the semantics of the idea from the language. This wouldn't necessarily be a problem, except these metaforms are now connected to their underlying semantics only nominally, having been denuded of their underpinning structure. Consequently, it makes determining semantic mappings that much harder because the complexity and nuance of the concept is now hidden.

Another point of interest is that each of the above efforts focused on identifying patterns from the ground up, but were limited to a very small pool of FOL ontologies. While it is definitely useful and necessary to also incorporate a ground up approach, given the small sample size, many patterns may be missed. Additionally, the unstructured development of these patterns may result in awkward navigation as users try to identify the appropriate template / metaform. A benefit of utilizing mathematical logical structures is that their behaviours are well understood and characterized, allowing for the development of a natural hierarchy.

Moreover, only the EZPAL effort attempted to develop an interface that would be more natural to users, whereas Staab et al required users to identify the appropriate metaform based on the predicate name. Visualizing and communicating the essence of the axioms to designers is an area that received little consideration in the above examples. However, this is not to say the visualization of logic languages has not been an area of consideration, though they are not the interest of this work; as noted earlier, it is increasing the accessibility of semantics, not the syntax that is being pursued here.

It is important to note that textual representations often evoke a linear understanding of the data, whereas diagrams and pictures may allow a more holistic representation (McLuhan 1964). Indeed, flow charts and graphs are often employed to provide a more accessible representation of data (Smith 1977). However, interpreting and understanding such representations is also a learned skill (Petre & Greene 1993). Some arguments presented by Smith in favour of visual representations include:

- There is evidence to support the notion that abstract reasoning is pictorial in nature.
- The two- and three-dimensional nature of visual reasoning may be more efficient than linear verbal language.
- Visual organizations are an extremely efficient "chunking" method to increase the capacity of short-term memory.
- Thought operations may be transformations of images.

- Images are usually analogous to tasks, while linear languages are mostly Fregean.[1] Analogical systems are more accurate than Fregean systems.
- Some experiments indicate that the mind is capable of storing sensory data in great detail. (Smith 1997 – adapted from Kremer 1998)

Indeed, Charles Peirce, one of the pioneers of formal logic observed:

> *I do not think I ever reflect in words: I employ visual diagrams, firstly because this way of thinking is my natural language of self-communion, and secondly, because I am convinced that it is the best system for the purpose [Ms 620, p. 8]* (Roberts 1973, p. 126)

Many people have investigated and developed a taxonomy of visualization techniques, though perhaps the most dominant is the one created by Myers (Myers 1990). Several of the relevant terms he identifies are included below:

- *Programming* – "a set of statements that can be submitted as a unit to a computer system to direct the behaviour of that system"
- *Visual Programming* – "a system that allows a user to specify a program in a two-(or more)-dimensional fashion."
- *Program Visualization* – "graphics [are] used to illustrate some aspect of the program or its run time execution"
- *Visual Languages* – Encompasses both Visual Programming and Program Visualization
- *Example-based Programming* – "systems that allow user to use examples of input and output data during the programming process." This can be either (Myers 1990)

[1] A Fregean language is one where the symobls bear no analogous relation to the task they are describing.

Visualization forms that may be used in representation include:

- Flowcharts
- Flowchart derivatives
- Petri nets
- Data flow graphs
- Directed graphs
- Graph derivatives
- Matrices
- Jigsaw puzzle pieces
- Forms
- Iconic sentences
- Spreadsheets
- Demonstrational
- None (Myers 1990)

2.4.1.3 Relevance

In ontologies, the most common form of visualization takes the form of graphs, where elements are nodes and predicates and functions are edges. Similarly, models of the axioms may also be represented in graph form, Hasse diagrams[2] being a notable example, where again nodes refer to elements and edges to predicates.

Several tools have been developed to visualize ontologies, including Fresnel for RDF (Pietriga et al 2006), Jambalaya (Storey et al 2002) for the Protegé ontology editor (Crubezy et al 2005), and Spectacle (Fluit et al 2003) which clusters instance data according to their membership in the ontology class hierarchy.

As the purpose of this thesis is to provide a hybrid ontology repository / upper level ontology for knowledge patterns, visualization here consists of determining what form best facilitates and communicates the essence and behaviour of the logical structure – i.e. their models. Romain and Cox identified the following points to bear in mind when determining the appropriate form of visualization:

- *Scope* – what aspect of the program is visualized?

[2] Hasse diagrams are graph like structures used to illustrate partially ordered sets. The convention is that edges are transitive, and elements "above" and connected are "greater than or equal" to those elements below.

- *Abstraction* – what kind of information is conveyed by the visualization?
- *Specification method* – what mechanisms does the animator use to construct the visualization?
- *Interface* – what facilities does the system provide for the presentation of visual information?
- *Presentation* – how does the system convey the information? (Romain & Cox 1993)

The above must be taken into consideration when designing the interface and visualization forms for the ontology repository. It should be noted that this first implementation only incorporates partially ordered sets, which lend themselves naturally to representation via Hasse diagrams. Moreover, the approach developed and discussed in the following chapters primarily makes use of example based programming. The physical interface is limited to keyboard, mouse, monitor computer interaction, ultimately accessible via the internet. The software environment makes use not only of graphics, but also English language descriptions in an attempt to be as comprehensive as possible and accessible to the widest array of users.

The approach developed here differs from those discussed at the beginning of this section in that not only is the wish for users to utilize more expressive axioms, but to also gain a deeper understandings of the "essence" of the axioms. The two go hand in hand, as users are able to access the essence of axioms by seeing them manifested via concrete examples in models. Furthermore, by simply weeding out unwanted models and accepting desired models, a process has been developed that can then derive FOL axioms that define an ontology designer's intended relation or function. Users may construct their own pictures, alter or pick pictures from a pre-selected set of models, and interact with the software environment in a graphical-textual manner by answering a series of simple questions to arrive at the appropriate set of axioms.

Given that this thesis aims to provide "logic Lego blocks" to the ontology community at large, the criteria of extensibility and reusability gain paramount importance. A

discussion of these two requirements at this scale (trans-disciplinary ontologies) gives rise to two terms, "upper level ontology" (ULO) and "ontology repository."

2.4.2 Upper Level Ontology (ULO)

A ULO is an ontology that serves as a basic starting point for more specialist ontologies. Recalling that an ontology is supposed to describe the shared understanding of a variety of people within a domain and given that almost no domain exists in a silo, there will often be an overlap of terms or concepts. Many ULO's develop a concept hierarchy, based on a few basic concepts, of which everything else is a specialization of. The taxonomic bias of "is-a" and "contained-in," also encourage representing domains in such a hierarchical manner. This idea is well illustrated by WordNet (a lexical ontology of the English language), where every word is seen to be an eventual "hyponym" of one of seven "fundamental" or "primitive" concepts (Fellbaum 1998).

Depending on the degree of specificity within a domain, it has been useful for ontology designers and users to name these systems as either Upper Level Ontologies (ULO) or as Domain Ontologies. The main driver behind an ULO is that it defines basic concepts that more domain specific ontologies may reuse. For example, a ULO might have the definition of Time and Events that more specific Physics and Biology ontologies might reuse. What exactly constitutes a ULO is somewhat ambiguous, for while it may be more general than some domain, the given ontology may be a more specialized manifestation of another.

While this method of trying to identify the most general, primitive or fundamental concepts has proven tremendously useful, it also gives rise to a variety of problems. Firstly, as Hayes has noticed, there are at least 14 different ontologies of time; which is the *right* one (Hayes 1996)? Is there uniquely, a right one? How might are these different time ontologies related to one another?

Secondly, as there is little agreement over which concepts *are* the most fundamental or general, it is difficult to translate between competing ULO's because they reflect the different biases and perspectives of their creators. For example, does math give rise to communication, or vice-versa? Is that a question whose answer would be worth pursuing? Several well known ULO's include SUMO, BFO, DOLCE, GFO, and CYC which provide such a concept hierarchies for other ontologies to reuse (Lenat & Guha 1990; Niles & Pease 2001; Gangemi et al 2002; Grenon et al 2003; Herre et al 2006). However, with the exception of the latter, very little logical axiomatizations have been provided for many of the concepts contained within the ULO. The degree of FOL proliferation in their "child" ontologies is even less overwhelming.

2.4.3 Ontology Repositories

The second term mentioned at the beginning of this section was "ontology repository." As the name suggests, these are often libraries of ontologies that have been catalogued, usually within a unifying theme or framework. One of the first repositories was the Ontolingua project at Stanford (now seemingly defunct) (Farquhar et al 1996). A relevant impediment in the greater utility of that repository was that many of its participant ontologies did not contain very precise axioms for their terms. Consequently, it was unclear how terms from different ontologies might interact as their semantic meaning was only nominally encoded in the ontology.

However, in the development of Ontolingua, many important considerations about maintaining and collaborating across a diverse array of ontologies and domains were articulated (Fikes & Farquhar 1999). A more successful ontology repository is the Open Biomedical Ontologies (OBO) Foundry, which seeks to insure that "a core of [its catalogued] ontologies will be fully interoperable, by virtue of a common design philosophy and implementation, thereby enabling scientists and their instruments to communicate with minimum ambiguity" (Smith et al 2007; OBO 2008). The actual delineation between what is an ontology repository and what is a ULO is somewhat blurred here, as OBO serves as both. The key is that OBO provides a forum where

conflicts that may arise among its catalogued ontologies can be explored and if possible resolved. In contrast, many ULO's restrict themselves to providing a top-down core that others will bifurcate and extend for their own needs. A downside to OBO ontologies are that they are mainly written using OWL and as documented above suffer from its accompanying expressive deficiencies.

Knowing which ontology is best suited to accommodate a problem is one looming challenge. Navigating amongst these and sorting them in an "intuitive" manner is another. Noy et al, identify the following requirements for ontology repositories:

- Ontology Summarization
- Rating of Ontologies
- Online Graphical Browsing
- Multiple-Ontology Search
- Ontology Mapping and Alignment (Noy et al 2004)

Ontology summarization necessitates a succinct description of what is represented in the ontology. It functions as an abstract type description meant for human level understanding. Rating ontologies serves to give users a sense of the quality and utility of the target ontology. Graphical browsing is an area almost all ontologies lack sorely today. Too often, the ontologies are presented only as flat text files which are tedious and unwieldy to navigate. Multiple-ontology search is a requirement if all these separately developed ontologies are to actually fulfill their promises of reusability and (reasonable) integration. Finally, ontology mapping and alignment is required to ensure that users know how each ontology in the repository is connected to another, and to validate consistency.

The ontology being developed in this work doesn't correspond to any particular domain or subject (although ostensibly, it is rooted in logical structures). Neither is it solely an ontology repository. It also functions as an upper (lower) level ontology in that it provides these snippets of logical structure to any ontology that wishes to plug in.

Similarly, it serves as an ontology repository as it catalogues the various uses of domain terms from the target ontologies, and serves as a base point where users may refer to or find other ontologies using similar concepts, but perhaps with different behaviour properties.

2.4.4 Metaphors

One of the main functions of the ontology repository is to serve as a logical substructure repository. What allows one to reuse these structures across diverse domains is the concept of metaphor. While the degree of influence and pervasive use of metaphor in our cognition is an intense field of inquiry and debate, it is uncontroversial that one way we employ metaphor is to map clusters of attributes from one concept to another.

Arguably, the very act of knowledge representation involves using detailed metaphors to represent complex concepts. For example, as Lakoff and Johnson note, in defining a material resource, we make use of the fact that it is:

> a *kind* of a substance
> can be *quantified* fairly precisely
> can be assigned a *value* per unit quantity
> serves a *purposeful* end
> is *used up* progressively as it serves its purpose (Lakoff & Johnson 1980, p65-66)

Each of these can be interpreted as describing relations – kind, quantity etc. – between material resource and some other object. While this type of metaphorical representation pervades the design of ontologies and most consensus based knowledge efforts, it is another type of metaphor, notably the structural conceptual metaphor that serves as the basis for the ontology repository. The structural metaphor works in the following way: imagine concept *A* has properties *(a1, a2, ... , an)* associated with it; similarly, concept *B* has properties *(b1, b2, ..., bm)*. The structural metaphor in this instance might map a1 to b2, a2 to b6 and so on, thereby mapping a substructure (occasionally, up to complete

isomorphism) of A to B. Thus when one says that "an atom is like a solar system," they are mapping those attributes of atom to solar system that follow the same logical scaffolding.

As Steve Pinker points out:

> *Scientists constantly discover new entities that lack an English name, so they often tap a metaphor to supply the needed label: selection in evolution, kettle pond in geology, linkage in genetics, and so on… As scientists come to understand the target phenomenon in greater depth and detail, they highlight aspects of the metaphor that ought to be taken seriously and pare away the aspects that should be ignored… The metaphor evolves into a technical term for an abstract concept that subsumes the target phenomenon and the source phenomenon.* (Pinker 2007, p257-8)

What this repository does (in its capacity as an ontology design aid) is turn the notion captured in the above quotation on its head. Ontology designers may have a term which they wish to define, yet the logical implications of how the term actually behaves might not be readily apparent. By engaging with the ontology repository, the closest approximation of logical axioms as stored in the repository would be located which describe the relation under consideration. In this way, instead of mapping metaphors en masse, and then pruning the ill fitting components, each mapping is provided bottom up, axiom class by axiom class.

2.4.5 Semantic Mapping Heuristics

In an attempt to address the issues of reuse and interaction among ontologies, researchers have developed the following heuristics to generate automated mappings:

- Mappings based on concept names (Hovy 1998)
- Identifying matching paths by interpreting an ontology as a graph (Noy & Musen 2003)
- Using instance data to identify matching concepts (Stumme & Maedge 2001)
- Developing logic isomorphisms based on information flow (Kalfoglou & Schorlemmer 2003)
- Probabilistic methods based on instance data (Doan et al 2002).

The purpose of this thesis is not simply to develop a semantic mapping tool, but to create an environment where logical structures may be identified and reused with ease. Some work has been done on "bottom-up" or structural mapping – ONION, MAFRA and FCA-Merge – the first two require the introduction of similarity, while the latter aims to develop a single merged ontology. In contrast, the focus here is limited to developing semantic mappings based on the underlying ontology structure, namely finding and matching equivalent or at least maximally consistent sets of FOL axioms. The semantics captured are contained in the logical behaviour of concepts, not necessarily in the lexical names ascribed to the predicates.

As a result, in the context of CLIF, semantic mapping is significantly facilitated, as the behaviour of elements *are* generally well articulated since much of the intended semantics is captured in the language. What is being tested is the logical consistency of predicate / function names to one another. This becomes a relatively straightforward affair if logical building blocks have been identified, then allowing automated theorem provers such as Prover9 (McCune 2003), Vampire (Riazanov & Voronkov 2001) and E-Prover (Schulz 2004) to test one set of axioms against another.

One advantage of this approach is that it renders redundant the problem of alternative formulations of the same axioms, as the theorem provers are able to establish consistency. Equivalence is only attainable in the ideal case, when the models of each ontology are isomorphic. Finally, allowing these types of mappings to be made explicit greatly reduces the ambiguity found in many ontological manifestations.

2.5 Summary

To recap the topics covered above, this project is motivated by two problems in the ontology community:

- Ontology Design (articulating axioms)
- Semantic Mapping

To address these problems, inspiration is derived from metaphors, to map not high level "foundational" concepts, but to communicate in the logical underpinning that reasoned thought employs. Consequently, a hybrid ontology repository / ULO has been proposed, which is close to, but not quite an open repository. Guidelines for adding and modifying the repository are developed in the following chapters.

Additionally, two algorithms have been developed to address the above two problems. One algorithm serves as an ontology design tool, allowing designers who may not be familiar with FOL to nevertheless express their ideas using FOL axioms. As noted above, not too many ontologies use FOL, relying on less expressive DL's, which it is conjectured is a large impediment to greater proliferation and use of ontologies. The second algorithm permits efficient automated semantic mapping based on the ontology repository.

Both algorithms rely heavily on work done by others (in the case of posets, mathematicians, logicists and computer scientists) in identifying useful and interesting classes of axioms. These are the theories which populate the repository, and guide the two algorithms. Eliciting and keeping track of additional axiom classes that may frequently arise but have not yet been cataloged also provides a beneficial feedback to the research community, highlighting structures to be investigated further. The following chapters detail the manifestation of these ideas.

Chapter 3

Ontology Repository Design

3.1 Ontology Repository Architecture and Design

While the repository will eventually encompass theories at any level of abstraction for a variety of domains, we first seek to develop FOL ontologies for each grouping or taxonomy of logical structures identified as interesting by philosophers of logic, mathematicians and computer scientists. Given our choice of language, we are looking for CL expressible theories (of which there is abundance), including but not limited to orderings, groups, symmetries etc. In principle, any grouping of logical structures that appear in use is a candidate for a theory that is also represented in the repository. For the purposes of this thesis though, the scope is limited to those theories that are well understood and defined. Once each of these theories have been identified and represented as an ontology, they will be put to use and made accessible via an ontology repository.

The repository provides a framework in which to collect, organize and link theories to one another. It consists of a series of layers and hierarchies as illustrated in Figure 3-2. One might note a similarity to an inverted traditional upper ontology but with a

significant difference. Instead of promoting a particular interpretation of reality, the repository beings not by defining "fundamental" concepts such as time. Instead, the lowest layer consists of those theories as specified by mathematical logicians and computer scientists - axioms which identify relations such as partial orders, symmetries, groups, geometries, etc. These determine structures which are both well understood and reused again and again in disparate domains. They serve as a foundation layer for the repository because these theories capture relations and patterns purely about the abstract syntax of the language.

Moving upwards in the repository yields layers of abstraction that are less generic. One way of connecting layers is via representation theorems; an alternative might to be to connect the layers via mapping axioms. At this point in time, it is unclear which is superior. However, irrespective of the layer linking mechanism, the second layer corresponds to the traditional domain of upper ontologies. Yet it should be reiterated that the repository does not assert the primacy of one upper ontology over another, it simply collects them as represented by CLIF. Thus the lowest level contains of theories for which no representation theorems exist (that is to say, they are not expressible as some other logical theories). The third layer consists of theories that are two representation theorems away from the lowest and so on.

Guidance from the traditional upper ontologies (general domain ontologies in Figure 3-2) drives the selection of which mathematical theories are of interest to include first in the logic layer. Similarly, there is the brute force bottom-up approach, which involves developing ontologies for as many first-order theories in mathematical logic. One might prioritize inclusion by scanning literature to determine the frequency of use of varying theories. The two approaches in tandem provide adequate cover to enable the following potentialities. It is a dual top-down and bottom-up approach.

By modularly collecting these blocks in a centralized repository, we develop a hybrid upper ontology cum repository. The repository functions as an inverted upper ontology, as the base layer contains not concepts such as space and time. Rather, the underlying

base layer consists of the mathematical theories that are isomorphic to structures in axiomatizations of space and time. Our repository then becomes an "upper ontology" where one may reuse different ontologies of space and time.

The repository serves to first catalog the different classes of logical axioms in a unified structure. The term "Core Hierarchy" is used to refer to different types of theories present at a particular layer of abstraction in the repository. Each hierarchy represents a map of non-conservative extensions of a particular relation or function. In general, explicit mappings do not exist between different core hierarchies; they function as silos within themselves. The only stipulation is that no theory in any particular core-hierarchy can share a non-conservative extension with another theory in another hierarchy. The preceding requirement entails that great care be undertaken when constructing the hierarchies.

In general, to establish the relation between two modules, one may either cite an existing proof in literature (as done for the modules in this version of the repository), or develop one's own proofs. Similarly, to show that two modules, A and B are not extensions of one another, if a proof is not readily available, one needs two counterexamples; one to show that there is some A that is not a B, and another to show that there is some B that is not an A. Due to the stipulation that no two core hierarchies share a non-conservative extension, when searching across the different hierarchies, the space of possible interpretations is limited, resulting the union or disjunction of the set of concepts. This point shall be discussed further in the ontology design tool chapter.

While the repository serves to collate the different core hierarchies, it must exist within a framework to aid ontology design and perform semantic mappings. Its likely guise will be that of a web accessible service, tied to a "sandbox" applet, automated reasoners and model generators. The full architecture of the resultant software artifact may be found in Chapter 6. The three main components of the service are listed below, while Figure 3-1 provides a high level view of the architecture of the environment.

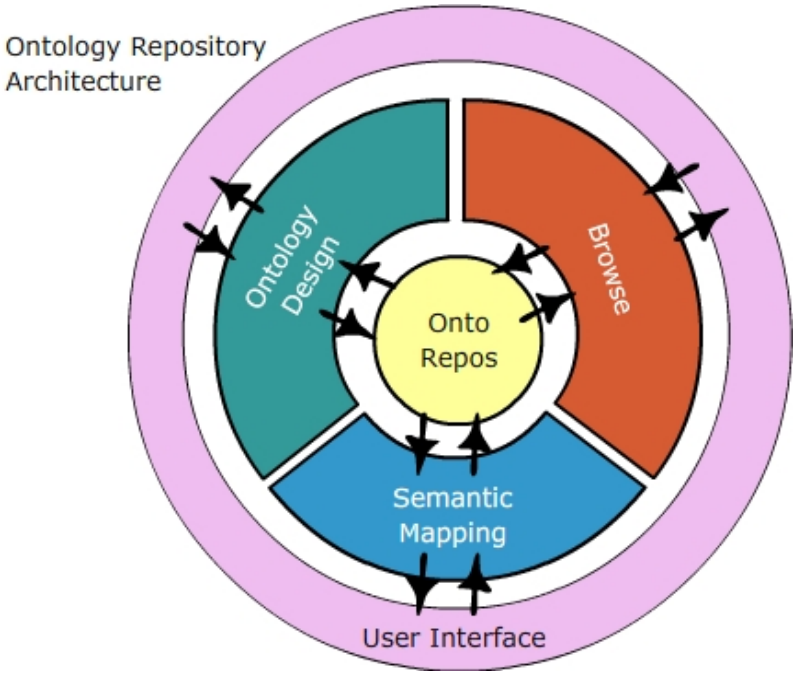


Figure 3-1 – Repository Architecture

A user interface mediates ones interaction with the environment, which itself consists of three elements: being able to navigate the repository (to make changes, add new elements or simply see what is in it), using the repository to conduct semantic mappings or using the repository to design an ontology. Each of these "services" requires some functionality as illustrated below.

- Logical Structures Core (Ontology Repository)
    - Different Layers of Hierarchies
    - Navigable Read Interface (view axioms, diagrams, similar to walkthrough)
    - Navigable Write Mode (ability to add new poset sub-classes and axioms)
    - Data provenance (history of changes + reasons)
    - Guidelines (description of potential problems / key terms - i.e. conservative extensions etc.)
    - List of examples
        - External links to uses
- Ontology Mapping Module
    - Mapping Engine

    - Mapping Interface
    - Integrated Automated Theorem Prover
- Ontology Design Module
    - Mini Tutorial
    - Algorithm – Interactive Example Programming
        - User Generated Models
            - Automated Theorem Prover
        - Axiom Generated Models
            - Automated Model Generator
        - Axiom Generation
    - Eliciting Novel User Axioms

The logical structures core is what will drive the other two modules (and be browsable). It must provide clear guidelines for the maintenance and expansion of the repository, as well as creating a community where users may contribute novel classes of logical structures. The two modules correspond to either semantic mapping, or aiding users in defining their relations.

3.2 Issues in Logical Representation

The primary conduit enabling the ontology design and semantic mapping is the repository of cataloged logical structures. Each group of structures, are referred to here as *Core Hierarchy*, since for each a containment hierarchy may be developed that links their axiom classes together.

To facilitate a practical environment for axiom generation, it is desirable to minimize the search space for potential axioms. Moreover, since the repository hopes to reuse work done by others, it draws on published explications of classes of axioms. Often, classes of axioms represent non-conservative extensions of one another, naturally leading to the containment hierarchy mentioned above. While this thesis implements one such ontology

for partially ordered sets, this approach may be extended to any theory that is expressible in CL.

By expressing these classes of axioms in a language such as CLIF, each core hierarchy represents a map to guide both the derivation of axioms and the semantic mapping process. The least constrained theories are located at the "top," while (possibly non-conservative) extensions refine the relationship in question, reducing its accepted model space. Every theory that is an extension of another, is considered "stronger" than all those above it in the hierarchy, since more can be proven with that theory than the others.

Clearly however, there is no guarantee that any finite or static number of core hierarchies would necessarily cover all the axioms a potential ontology designer would desire. In this regard the repository is limited, and it should be noted that the decision was made for the first iteration of the repository to initially incorporate only well understood theorems. Additionally, in contrast to ULO's, the claim is not being made that these classes of structures are the only foundations for possible ontologies. Rather, we acknowledge there is room for novel classes to be added, expanding the vocabulary of the repository. This also helps reduce some of the clutter and inaccessibility some may feel when working with FOL. It is hoped that in the future, the ontologies stored in the repository will reflect not only those noted as of interest and catalogued by mathematicians, but novel classes that have their roots in how people actually use FOL in practice. Moreover, different layers of abstraction may also be added, for example, ontologies covering space, time, parts etc. this is illustrated in Figure 3-2.

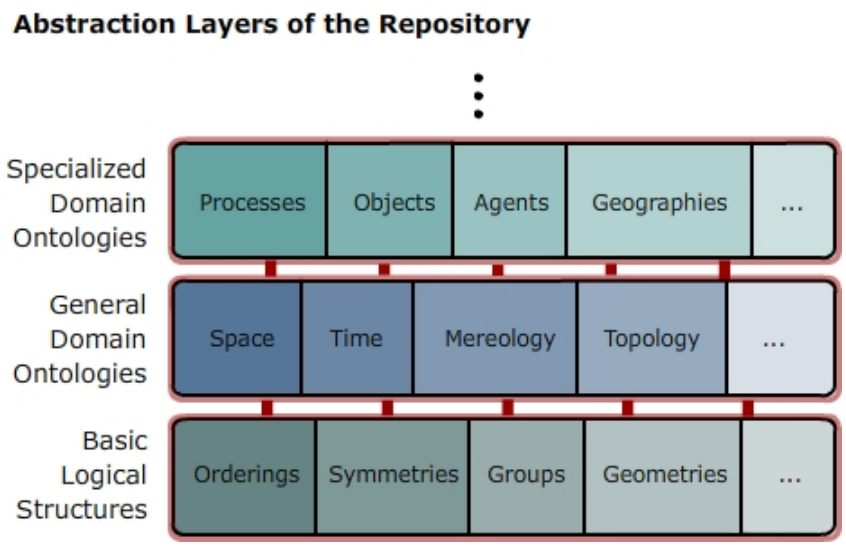


Figure 3-2 – Repository Layers

Gathering and storing such core hierarchies in the repository necessitates investigating the following issues:

- Which machine readable language to use?
- If multiple equivalent axiomatizations for a concept exist, which to use?
- With an eye on modularity and reusability, how does one accommodate extensions of concepts?
- How do non-conservative extensions affect choices with namespacing, importing modules, or defining new classes?
- What ought the guidelines be to aid future contributors to add or modify existing hierarchies
- What other issues must be considered when maintaining a large, changing repository?
- How might one make an ontology accessible to a human viewer?

Each of the above points shall be discussed at length in the following sections.

3.2.1 Language

The language chosen to represent the axioms for poset classes was the Common Logic Interchange Format (CLIF), as documented by the ISO (CL Standard 2007). The reason behind the choice of CLIF is multifold:

- it has adequate expressivity (FOL complete, *plus* the ability to quantify over relations in a restricted context and introduction of sequence variables)
- it is being supported by the ISO
- it has large community for support and documentation

Many logical theories require a language that has expressivity beyond containment to be of use; often they need a language that has the capability of representing axioms in first order logic. CLIF is a language that both has active researchers contributing to its growth and design and affords adequate expressivity. Additionally, given that the aim of this ontology repository is to be as widely used as possible, the growing popularity of CLIF and its endorsement by ISO was an influential motivator in its choice as the utilized language.

3.2.2 Axiom Expression

In the literature of mathematical logic, depending on who is doing the writing (computer scientists, logicians, etc.), axioms are often expressed in superficially different but equivalent ways. As noted in the previous chapters, one can equivalently use a variable, function or relation to point to the same *thing* in the world. One concrete example is found in posets, where one can use either functions such as “meet” and “join,” or relations such as “minimal upper bound” (MUB) and “maximal lower bound” (MUB). Both exploit a relatively straightforward abstraction of properties of the “less-than-or-equal-to” relation which underpins the logic of partially ordered sets.

Each different formulation has its own strengths and weaknesses, largely motivated by the domain in which they were articulated. Often, issues such as these, the lack of guidance in terms of which articulation one should take and whether one is "superior" to another, impede a wider adoption for FOL ontologies. As a designer, it would be presumptuous to elevate one type of formulation over another. The aim of this repository is not to advance a particular FOL theory or style, but to be as accessible and malleable to as many different people and contexts as possible.

As a result, whenever multiple articulations of the same theory exist in literature, all should be represented and stored in the repository. Moreover, a simple mapping would allow one to link the two conceptualizations to one another easily, since they represent superficially differing accounts of the same underlying theory. Again, using posets as an example, the concepts of meet and join do not necessarily exist in all applications, whereas relations such as MLB or MUB which describe the same phenomena might. So long as the underlying mapping is available (and for structures drawn from mathematical logic, they almost always are), it is trivial to store all forms in the repository. While this may place an additional burden on the repository contributor, the versatility it affords far outweighs that initial effort.

For the purpose of this ontology repository and the current core hierarchy implemented, axioms for all the poset classes will be expressed in both conceptualizations where applicable. This redundancy should allow designers to use whichever abstraction is more "natural" for their domain of discourse. As noted above, although the concepts of meet/join might be awkward in one field, they may follow naturally in another.

One final complication introduced by this design decision occurs when updating the repository; care must be taken to address changes in both versions simultaneously. Since the underlying containment hierarchy is simply mirrored using different terms, it is easy to make note of these changes, and this does not pose a serious problem. As before, this additional constraint seems well worth the cost of increasing the accessibility of the representations.

3.2.3 Modules, Classes or Namespacing?

Another driver of the repository is that it be amenable to extensibility. Deciding which set of tools to use to manage extensions is an important one, and depends on the type of extension being made.

Since the language used in the repository is CLIF, we must note how "classes of axioms" are expressed in this syntax. In CLIF, classes of axioms may be demarcated by using the "cl:module" string, where a module is named, and within it, the relevant axioms are stated. Combining modules together involves using the "cl:import" term, and listing the name of the module one wishes to import. Thus if one had defined two modules:

(cl:module (poset)
    …AXIOMS)
and
(cl:module (lattice)
    (cl:import (poset))
    …L_AXIOMS)

Equation 3-1

the resultant axioms for the *lattice* module would be the union of AXIOMS and L_AXIOMS. This type of extension corresponds to the module approach noted above.

Using posets as an example, it should be noted that lattices are non-conservative extensions of posets, meaning that one can prove novel properties about the relation, "lte" (less than or equal to) being defined.

Another alternative in extending a theory would be the "classes method," where one defines a new relation for each new extension. Thus "lte" would originally be defined in poset, and an extension of that relation, say "lattice_lte" would be defined as a novel

class and relation. The import statement would still be used as above, however the new lattice axioms now only apply to the "lattice_lte" term, and a new statement such as

---

(forall (?x ?y) (if (lattice_lte ?x ?y) (lte ?x ?y)))

Equation 3-2

is required to link the two to one another.

Namespacing differs in that one may use the same term name, however its definition is only local. In practice, it looks very similar to the class method. So instead of "lte" and "lattice_lte", we might have "lte.poset" and "lte.lattice," where the additional string after the period indicates which context (namespace) of lte is being referred to.

One final consideration not investigated in depth in this thesis, is the ability of CL to quantify over the relation types. Whereas Equation 3-2 defines classes, the non-segregated dialects allow us to make the following assertion:

(forall (lte) (if (lattice lte) (poset lte)))

Equation 3-3

thereby noting that any *less-than-or-equal-to* (lte) relation which is a lattice is also a poset *lte*. This thesis has chosen not to capitalize on this added expressivity for two reasons. First, no freely accessible theorem prover supports these types of axioms. Moreover, most FOL applications also do not support such formulations. However, if in the future, the bulk of FOL expressible theories use non-segregated dialects, it would be a simple port to convert class linking statements to relation quantifying statements. The superiority of one of these methods in relation to another depends on the types of extensions being made, and is substantiated by the discussion in the following section.

3.2.4 Conservative vs. Non-Conservative Extensions

In general, as one progresses "down" a containment hierarchy, each extension of the previous axioms may be classified as either "conservative" or "non-conservative." Assume two modules, T_1 and T_2, where T_2 is an extension of T_1. A conservative extension implies that the addition of new axioms does not alter the meaning of constructs within T_1. In contrast, if T_2 is a non-conservative extension, novel theorems may now also be proven about T_1. Since in general, both types of extensions are possible for any given core hierarchy, we must accommodate the latter, more thorny extensions.

Using classes or namespaces to define the new senses of a relation ensures that each extension is conservative. Since each relation in every module in the hierarchy now has a unique identifier, changes in extensions of the model do not alter the "local" or sense-specific meaning of a given concept. While this allows users to locally modify their own interpretations without altering the meaning of more general relationships, it does lead to a proliferation and bifurcation of names. As the number of classes of interest increase, the requisite vocabulary also increases.

While the modular structure of axiom inheritance does not ensure conservative extensions of relationships, careful ontology architecture may maintain the same locality of meaning as achieved by creating a new name for each variation of a given relation. For example, let us consider the following example of a theory hierarchy and extensions as shown in Figure 3-3. As one moves "downwards," the particular theory T_i presents a non-conservative extension of those connected to and above it:

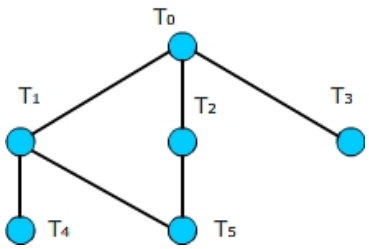


Figure 3-3 – Theory Extensions

There are an infinite number of scenarios one can construct involving interactions between two or more relations that are in some ways extensions or branches of one another. We can collapse these scenarios to four general cases of how the relations are used within two ore more theories, as illustrated in Figure 3-4.

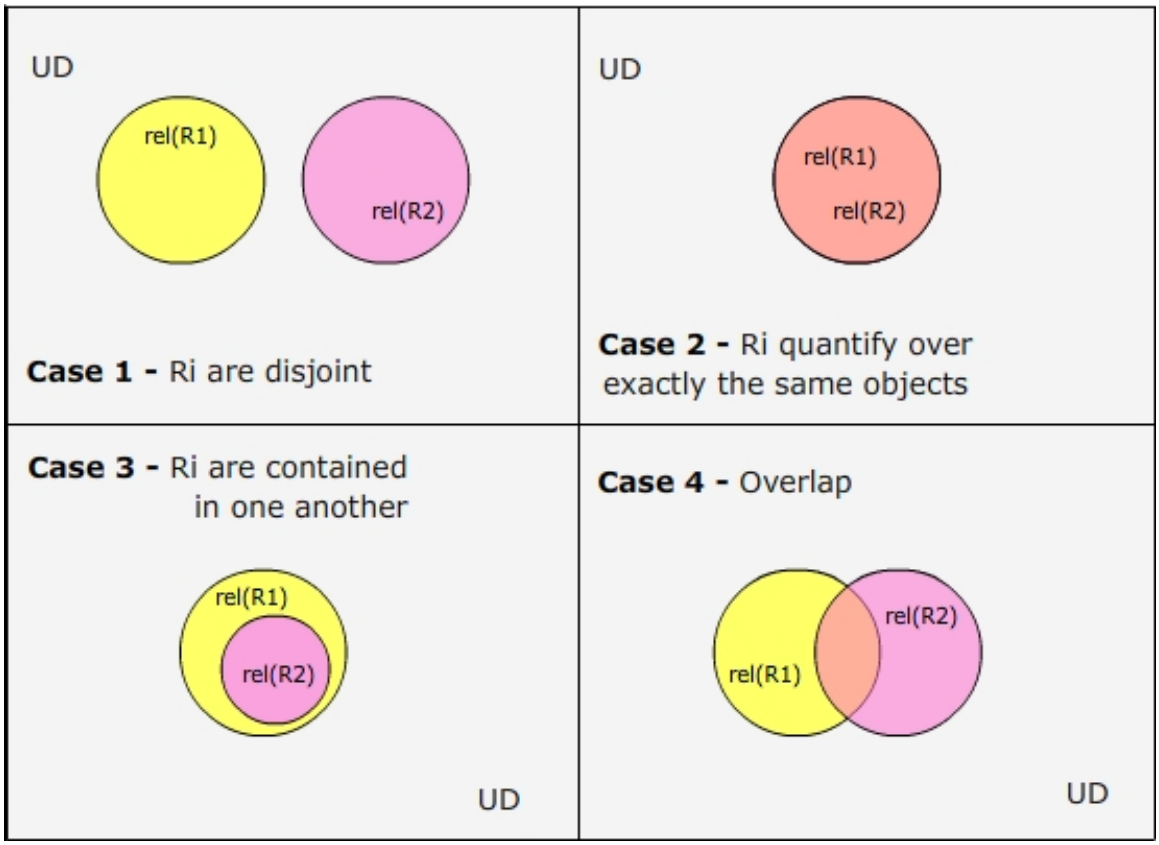


Figure 3-4 – Theory Overlap

Case 1:

Relations are disjoint in terms of objects in the UD that they quantify over.

This corresponds to the case of having implicit "sorts." Using only a modular-import architecture, one has two options:

1. Define novel modules keeping the two relations separate and insuring the two branched senses of R are never called in the same module
2. Introduces new unary relations corresponding to "sort-type", adding either one argument to (increasing the arity of) the relations being intermixed or adding an "and" term to attach the sorts to the relation when appropriate.

Classes or namespacing would instead use a distinct relation name for the sorts each R ranges over, thus bypassing potential conflicts in meaning.

Case 2:

Relations quantify over exactly the same objects in the UD.

In this case, the relations form an extension of one another, even if they previously branched from a more general R. In essence, a new R has been defined and any module further importing this one could only refer to the union of the R's. The new name approach would unsurprisingly have defined a new R for this interaction of axioms.

Case 3:

The relations are nested, such that R_1 ranges over D_1, while R_2 ranges over D_2 and $relI(D_1)$ is contained in $relI(D_2)$.

We again have two options:

1. do not use the R's in the same module, instead use two or more.
2. the same "sort" problem, except here the sorts are subclasses of one another. Necessary to explicitly restate in this module the existence of types of objects and the restriction of quantification over them by the relation. One would also need to increase the valence of the relation.

Case 4:

There exist come overlap in terms of the objects such that $relI(R_1) \cap relI(R_2) \neq \emptyset$.

This is the final permutation of the sort problem, and is addressed in the same way as those above. The Classes / Namespacing approach would have instead defined a unique name for this scenario. Moreover, since they share some interpretations, we can also conclude that T_1 and T_2 are consistent with one another.

One can construct additional cases by taking combinations of 1-4 across different numbers of relations. However, they each illustrate a basic issue in trying to accommodate these situations using only module imports. The (cl:import) phrase only captures Case 3, where one module is an extension of another, the other cases will need to be captured at the metalevel.

Extending relations using only module-import and using CLIF's language style would push the definition of new names "down" a layer, specifically to defining and including the "sorts" of objects which R is ranging over. In contrast, extending relations through namespacing / classes defines the new names at the level of the relation. However, the proliferation of names is aesthetically unappealing, particularly so when one considers how people actually use the relations being defined. Rarely do people say "this is a lattice partial order," rather, it is more natural to say "when we have a lattice, the partial order relation exhibits these new properties."

While it is possible to accommodate the module-import using Common Logic to quantify over unary-relations (the sorts), strict FOL interpretations would not be able capture this idea. It is noted however that this approach does seem to make a theory more accessible, since it explicitly defines the objects over which the different senses of R occur as opposed to implicitly referring to them in the creation of new relation names.

Limiting ourselves to the TFOL approach, the modular approach would require careful consideration on the part of the ontology designer if a relation applies differently depending on the sorts of elements it ranges over. At times, this might be awkward and

detract from the intent of the ontology designer, however it ensures that the relation always carries with it the explicit intended semantics. When using namespacing and class approaches, no such restrictions on the composition of the ontology. Whenever a new sense of *R* is required, the designer simply creates a new class or name for the concept, and includes a line similar to EQ-3-2 above to link the two relations to one another. It does however introduce problems in combining theories – is the resultant combination, (a) a union of the different relation names under a new name, or (b) is it the disjoint application of the different theories?

While the limitations of both implementations have been discussed above, the goal of the repository is to lay the foundation for a scalable resource which maximizes reuse. The namespacing and classes approaches are almost identical, except for the fact that the former emphasizes the link between the name of the module and a corresponding relation. In contrast, the latter accentuates the axioms that comprise the relation itself.

As noted above however, the proliferation of new names modifying the relation in use is rarely what is most "natural." What we would really like is the greater expressivity as afforded in EQ-3-3. We preserve the same relation name throughout all its applications, but explicitly, semantically, represent the connections between the modules. For the time being, these connections are stored at the met alevel and captured via core-hierarchy maps as in Figure 3-5 below. Consequently, in the implementation of this thesis, the module approach is taken.

3.3 The Repository as a Software Artifact

3.3.1 First Ontology in the Repository

Picking which types of logical structures to present and axiomatize in the first iteration of this ontology repository was a difficult task. While the current catalogue of poset classes and axioms is wide ranging, it is by no means a comprehensive assessment of all characterized FOL expressible theories of interest. At present, the proliferation of these mathematical and logical constructs outside of their respective domains, to real world

applications has been fairly limited. Every module included in the poset repository at this point in time is illustrated in Figure 3-5 below.

Moreover, new classes of axioms are being identified on a regular basis, though their practical applicability / utility with respect to the relations people actually use to describe concepts / constructs of interest is unclear. The current list of poset classes have been culled from three different implementations in mathematics and computer science. While the al appear in the same hierarchy and are all extensions of the partial ordering relation, they are referred to as:

- trees
- lattices
- graphs

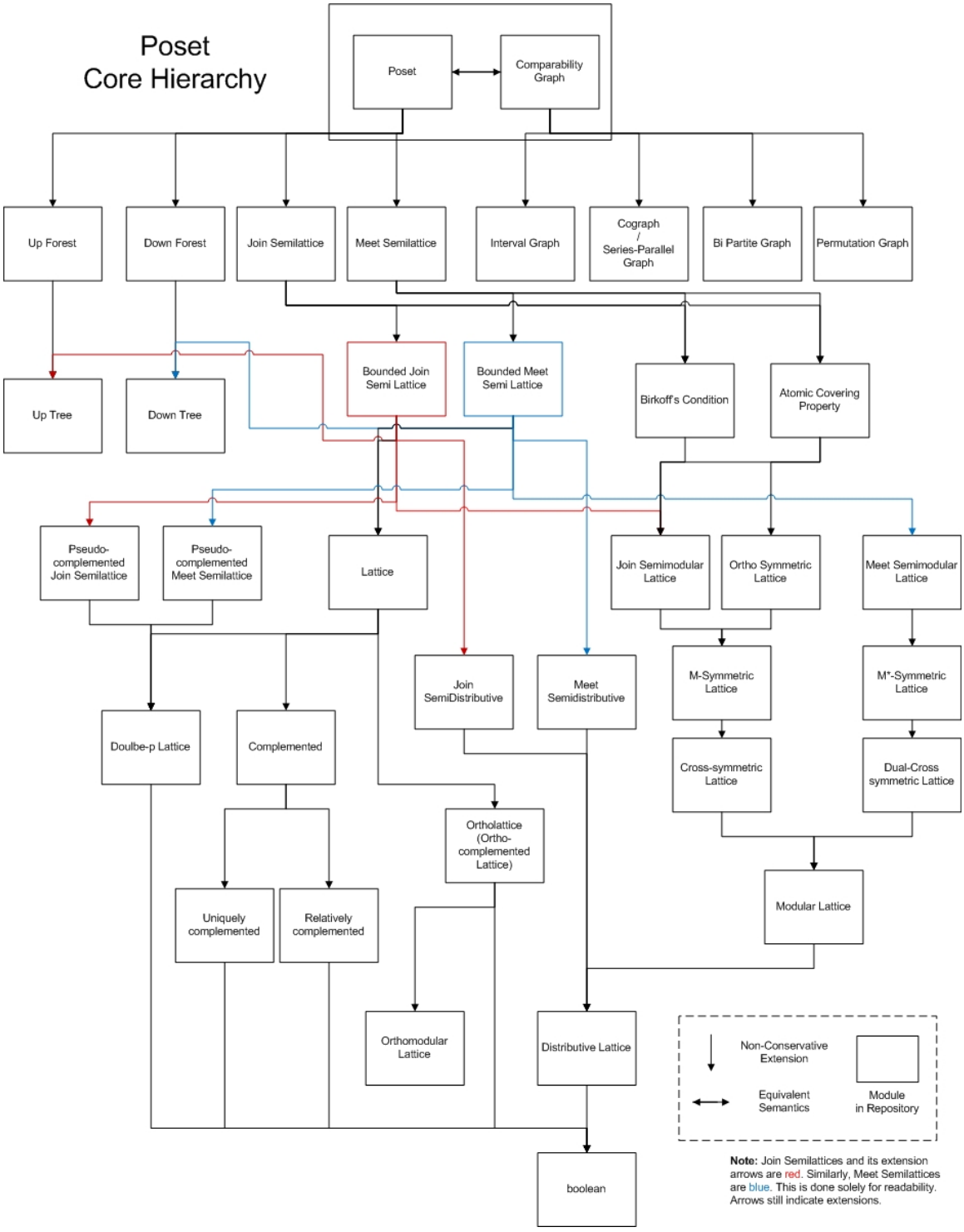


Figure 3-5 – Poset Hierarchy

3.3.1.1 Trees

Trees form arguably the most accessible classes of posets which are widely recognized by people not particularly familiar with first order logic. They correspond to many people's understanding of hierarchies and are found in the guise of directory structures on computers, or in the implementation of many library classification schema.

Four different types of trees have been catalogued in the ontology repository:

- trees that are rooted at the top
- trees that are rooted at the bottom
- forests that are rooted at the top
- forests that are rooted at the bottom

The axioms for each of these concepts are presented using only terms such as *less-than-or-equal-to* (*lte*), which in file directory language would roughly correspond to *is contained in*, and the notion of *root*.

3.3.1.2 Lattices

Lattices are another of the most commonly utilized and well characterized class of posets. The full list of catalogued lattice types may be found in Figure 3-5 below which represents the integrated hierarchy for posets. As illustrated, lattices form a well understood hierarchy of axioms, with the most constrained class of lattices located at the bottom, in the form of *Boolean lattices*. The choice of which lattices to include was largely influenced by the number of papers that cited each lattice type and their "popularity" in order theory. Several real world examples also make extensive use of lattice structures (see PSL's subactivity relation across atomic_activities), which further reinforced the choice of which lattice types to include.

3.3.1.3 Graphs

The number of graph types that correspond to posets are exceptionally numerous, though their differentiation often come down to applying seemingly esoteric refinements to graph axioms. As cataloged by (ISGCI 2008) there are hundreds of graphs that are specializations of comparability graphs, often defined in terms of particular forbidden substructures (ISGCI 2008). As seen in Figure 3-5, only five classes of graphs have been incorporated into the first iteration of the repository:

- comparability graphs (same as basic poset)
- cographs / series parallel graphs
- permutation graphs
- interval graphs
- bipartite graphs

Many of these graphs have implementations that have already been found in real world applications of the understanding of relationships. The design choice at this point in time was limited to these five classes of graph posets because they correspond roughly to the most utilized, understood and characterized types.

3.3.2 Navigable Write Mode

As stated earlier, the current implementation of the hierarchy includes only one core hierarchy, that of posets, which is itself a subset of the ordering hierarchy. Future contributors may wish to either extend existing hierarchies or add new ones. One of the main goals of the repository being developed is to serve as a type of logical structure metaphor resource, where the different underlying constructs that connect various disparate concepts may be accessed. However, one of the limitations of metaphorical mapping is that there does not always exist a one-to-one mapping between a pre-existing concept and a novel one to be described / identified. When providing axioms, only the most specific applicable axioms will be provided by the repository, which may in the end

lack some additional refinements that an ontology designer may desire. Since any particular hierarchy is limited by the ability of a contributor to identify and articulate axioms, support for introducing novel classes is of paramount importance.

At the same time, it is important to note that the repository being developed is *not* completely open; that is to say, not anyone can create new core hierarchies or modify existing ones. The integrity of the axioms is instrumental to their use, and the trust of ontology designers and semantic mappers. Thus, the creation of an account system for contributors to this repository is necessitated.

3.3.3 Community and Contributing

An account management system (not developed in this thesis), is required to allow changes to be made, and to keep track and save "sessions" for designers. In general, the following user classes are desirable:

- Guest
- User
- Contributor

Not everyone will want to login or register to use the ontology, so the Guest option is available. However, should an ontology designer wish to contribute back to the repository, then she is encouraged to register an account. Doing so would provide her with an opportunity to not only save "sessions" – i.e. pause the ontology design tool process and return to it later – but to also allow the repository to catalog and keep track of where the axioms are being used. This facet of the repository is discussed in greater detail in the List of Examples section below. Briefly, a user would be asked to provide name, email and a URI (if it exists) for their ontology.

Finally, some users may wish to extend or modify the repository, leading to the contributor user class. When one is logged in as a contributor, and enters the "write

mode", they are asked whether they wish to make a modification to an existing entity, or whether they wish to create a new entry. If they select modification, from the relevant navigation hierarchy (discussed in the proceeding sections), they may select whichever class they wish to modify, and once that has been "zoomed in on," the specific component of the class may be selected. An automated data provenance record is created with each new modification, with a roll-back copy of the previous entry created. At this point, the new entry will enter "verification," and will have to be verified by other contributors before it is accepted into the ontology repository. While the theories will be accessible, they will carry a warning that they have not yet been verified.

If a new entry is made to an existing core hierarchy, the contributor is also directed to the navigation hierarchy, and should be able to specify exactly where in the hierarchy they wish to define a new class of axioms. All three components of a class description must be complete before the novel class would be accepted to the repository core. In addition to requiring all three fields to be complete, novel entries to the repository will also require approval from two independent reviewers.

Again, it is important to stress that the imposition of reviews before the acceptance of modifications or additions to the ontology repository is that this framework is not a wholly open one. The integrity and rigour of the repository must be maintained to insure that the axioms produced are accurate and do not lead to unintended models. While no clear standards have been produced by the knowledge management community vis-a-vis how repositories ought to be maintained and extended, the model is clearly different from the one offered by Wikipedia. Similar to that project though, forums for discussion pertaining to guideline changes or fundamental alterations of the ontology repository design architecture should also be supported.

3.3.4 User Generated Axioms

One other component of the repository will be the ability to allow the repository to grow from bottom up as well. In general, we assume that an ontology designer's intended axioms will be:

$$I = T_i + O_i$$

Equation 3-4

Where I is their intended axioms, T_i are the classes of axioms that are stored in the repository and O_i are axioms that are external to the repository. The exact nature of the algorithm for extracting the appropriate set of axioms for a user's I will be discussed in the Ontology Design Aid chapter, however it does suggest the development of an additional feature for the repository.

Should the composition of O_i become available, this new theory might be a candidate for inclusion in the repository as an extension of some of the T_i's. For example, there might be a particular application where the user requires that the relation they are defining, R, be a tree, where each node has exactly three leaves. While these types of particularities are not presently captured in the hierarchies, they may be of utility to ontology designers in practice. Consequently, collecting these axioms when possible would increase the scope of the repository and the ensuing algorithms.

Including these novel sets of axioms in the repository intimates the establishment of a mirror of each core hierarchy which includes user defined axioms as well. When a designer engages the repository, they will be given the option to peruse the "original" hierarchies or the more extensive, but potentially thornier user enhanced hierarchies. This option is provided to ensure that there will be hierarchies which aren't too unwieldy or idiosyncratic, and where the precise behaviour and properties of each axiom class are understood. For one, the number of theories in the hierarchy which includes user-

generated content might be too large for the algorithms in chapters 4 and 5 to navigate effectively. Moreover, since one criterion for inclusion in the repository is that a theory be relatively well understood, this may not be the case for these novel classes of axioms.

One other approach to combat possible unmanageable expansion of the different number of classes in the user enhanced hierarchies may be to first collate these new axioms. Inclusion would then be subject to having them satisfy certain requirements before being presented in the hierarchy. Such a requirement points to perhaps having three different hierarchies for each type of logical structures one where it consists solely of theories taken from well known sources; another is an extension of that with *selected* user generated classes; and the third would be an extension of the original with *all* user generated axioms. Deciding which user axioms should be included in the middle mirror is difficult to judge a priori, without a body of in situ evidence. The most immediate measure might be a histogram, where if multiple users desire the same O_i, then that would suggest that those axioms should be included. Finally, it should be noted that both Users and Contributors may propose user generated classes of axioms.

3.3.5 List of Examples

One of the potential highlights of this site is that as users begin engaging with the repository, a catalog of defined relations linking to the accompanying ontologies where they are used shall grow. In this way, relations within a domain may be compared to one another, or relations may be compared to those that have similar logical structures.

The final non design aid / semantic mapping component of this service includes a growing list of ontologies that have made use of the ontology repository to either map their relationships or to define novel ones. Future designers should be able to more easily connect to other developers who may be working on similar issues, or who may be in a completely different field, yet have key relationships that exhibit the same logical behaviour.

This trans-disciplinary similarity and relationship mapping is the essence of the metaphors that this repository is attempting to document and facilitate. While the framework cannot make any predictions or conclusions beyond semantic or axiomatic mappings between two disparate relationships, it can provide valuable service in serving as a focal point where previously unexplored logical similarities between fields may be elucidated and if deemed worthy of exploration, pursued.

Two views will be utilized for the presentation of these links, organizing the external ontologies either by their subject heading (as determined by the ontology designer who used the repository to derive her axioms), or according to which classes of axioms the defined relations/functions correspond to. In this way, both similarities between relations will be highlighted and divergences in the understanding of concepts among intra-disciplinary ontology designers may be explicated. Links to the external ontologies will also be provided where possible, though this relies on inputs from users of the repository.

3.3.6 Data Provenance

Data provenance is one of the most important facets of the ontology repository, as it shows the evolution of the various poset classes and their descriptions. Based on research done by (Simmhan et al 2005), having a comprehensive and thorough data provenance component to any large collaborative project of this nature is an absolute necessity. As alluded to by Simmhan et al, provenance is used for:

- data quality
- audit trail
- replication
- attribution
- informational (Simmhan et al 2005)

In the context of this repository, where the integrity of the knowledge being stored and represented is of paramount importance, we must keep track of changes to classes. We

need to know who changed what, why, and what was changed. There are several elements of interest, changes to the classes of axioms, changes to the two algorithms, and changes to the guidelines governing the use and maintenance of the repository.

As noted before, whenever someone makes a proposed alteration / extension to a core-hierarchy, pending review, versioning data must be added. Any accepted addition of new classes of axioms would include the date of inclusion, the author's name, alongisde a new version of the hierarchy which now includes the novel class. For alterations of existing classes, it will be necessary to keep track of each pre-existing copy of the class, alongside standard provenance information.

Similarly, information on who has made changes, and what the changes were, need to be collected for alterations in either the algorithms or the guidelines of the repository. Each alteration should be stored in a new version, with previous versions being kept for reference. The following two chapters will now explicate the two algorithms for ontology design and semantic mapping.

Chapter 4

Ontology Design Algorithm

4.1 Motivation

As has been noted in an earlier section, the majority of ontologies published and available are varieties of "terminological ontologies," comprising of named concepts and a paucity of relations (usually restricted to subsumption, or some subset of FOL). While these types of taxonomies are certainly useful, their limited expressivity and consequently narrow scope restricts the applications that may be developed around them. Axiomatic ontologies which additionally describe the concepts being named and hence provide explicit semantics for them are less prevalent.

Specifying the actual behaviour of these concepts, especially in a machine-readable form often requires one to formulate axioms in a formal language (i.e. FOL). Unfortunately, many subject matter experts (SME's) and even some knowledge engineers have difficulty articulating their thoughts in these languages. These languages are often seen (erroneously in this author's mind) as too mathematical or too divorced from the reality

which they are attempting to describe. This problem is exacerbated by the often unwieldy syntax of such languages and further compounded by the lack of clear guidelines as to how one ought to write axioms.

The purpose of this tool is to allow those who have difficulty or are unable to communicate or articulate an idea in the language of first order logic to nevertheless be able to express their ideas in such a form. Instead of requiring familiarity with the grammar and syntax of FOL, agents must be able to conduct only the following two tasks. First, they must be able to develop at least one possible model of the world using their idea in a supported representation system (i.e. graphs). Second, agents must also be able to discern whether a particular model of the world is one they would accept or reject as a manifestation of their intuition.

4.1.1 Communication and Language

To revisit the source of this formulation, we must consider the nature and purpose of languages in the context of communicating the internal idea of an agent. Figure 4-1, illustrates a simplified version of the process of the transcription of thought into a language accessible by any other agent. It should also be noted that agents in the diagram below can be human, electronic or otherwise.

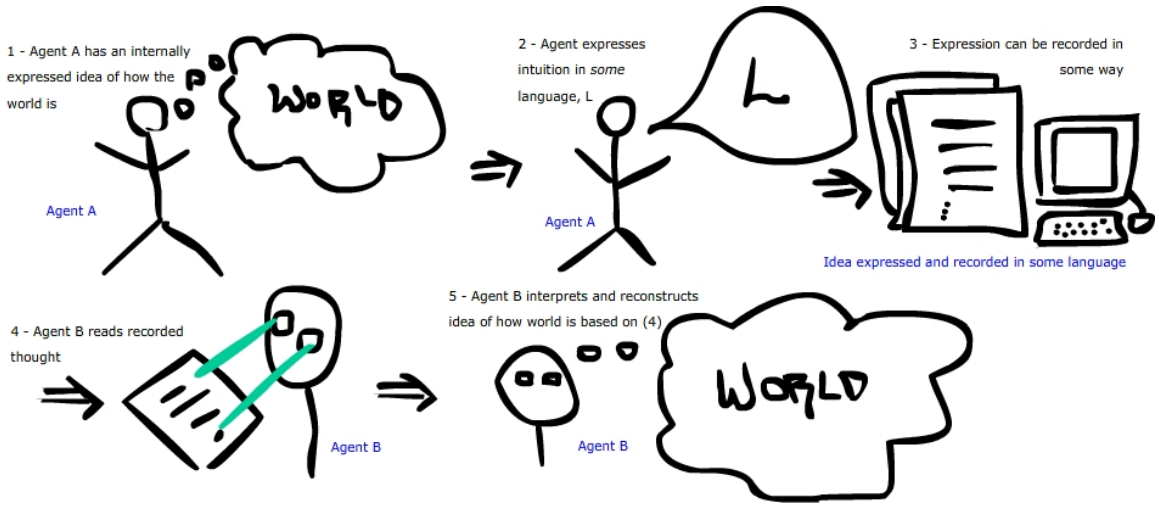


Figure 4-1 – Communication and Language

What an ontology designer would ostensibly like to achieve, is to communicate and thus express their internal understanding of the world in a form accessible to external agents. For agents poised to leverage the use of automation, this means expressing and capturing the internal intuition through some language. In the context of this thesis, we are considering the expression of an idea in a formal language and in particular, first order logic.

4.1.2 Expression and Models

In this environment, language functions as a mapping between our internal understandings of the world to an externally accessible format. It should also be noted that here, *world* is loosely defined as things that are relevant (to know, to acknowledge, to account for, etc.) The exact composition of what the world *is*, is left for others to explore. Such an incomplete list suffices for our purposes at the moment.

Given that we are seeking expressions of internal intuitions in the form of FOL, we can then make relatively precise, logically founded statements about the world. The result of any such articulation shall be a set of axioms, let us call A. Any set of axioms has associated with it a set of models. Models can be thought of as capturing some sets of truth in the world. Often, these sets of truth are manifested as particular instances of the world as reflected through the language used. Specifically, a model exists so long as there is an observer or other agent that assigns *meaning* to the symbols of the language. This process is similar to how equations used in physics or biology yield particular meanings as applied by a physicist or biologist. The agent correlates the abstract symbols of the language, strings, with literal things in the *world*. Thus for any set of axioms A, there exist a set of models M^A that are consistent with it. Similarly, any model may be converted into a set of sentences by generating the *complete diagram* for that model, where diagram is the mathematical term which is understood to mean the set of all positive and negative literals that are satisfied by *M*. For example, if I have a set of axioms for the relationship *beside* – a possible model might be 1-2-3-4.

4.1.3 Conventional Representation

Traditionally, if an ontology designer wishes to express an idea in a particular language, then she would have to learn to read and write in said language. For many people, this presents an additional burden that may also not be an intuitive or adequately timely process. Moreover, such efforts may also lead to the development of axioms which inadvertently allow a plethora of unintentional and misleading models if the designer is not sufficiently careful.

The reasons above partially contribute to the thus far limited proliferation of the more expressive first order ontologies. However, noting the correspondence between axioms and models, and leveraging the work done by logicians in cataloguing many first order theories, this thesis generates a novel ontology learning process. Ontology designers would be able to develop FOL axioms to communicate their intuitions, without needing to directly articulate their ideas in the syntax of FOL.

4.2 The Algorithm

To allow users to constructively and iteratively derive a set of FOL axioms to capture the essence of their idea, the interplay between two agents and the world they are trying to describe is exploited. Extending Figure 4-1, we make one important change, complete the circle and have an interactive communication between Agent A, the ontology designer, and Agent B, the ontology design tool. In theory, the discriminating agent who decides whether a particular model corresponds to their intuition need not be human, but such automation presents a whole load of problems outside the scope of this thesis.

Instead of requiring communication to occur through the medium of FOL, conversations are instead conversed by using models built on said language and simple vetting process of whether a proposed model is acceptable or not. The essence of the algorithm is illustrated in Figure 4-6 below. It is also illustrated via two use cases in section 4.4.

We can assume a priori that a user's intended axioms, *I*, are decomposable to:

$$I = T_i \cup O_i$$

Equation 4-1

where T_i are sets of theories that are present in the ontology repository, and O_i correspond to classes of axioms not yet represented. While we cannot search out equivalence between I and T_i given the existence of arbitrary O_i's, we can determine consistency between I and T_i. It is only in the ideal case where $O_i = \perp$, where equivalence maybe found. Otherwise, this algorithm is limited to finding only the T_i's in I, while remaining silent about the O_i's.

Additionally, because we are working in the world of models, and it is not practical to enumerate and present all possible M^0_i, the strongest claim we can make about the results of this process is that T_0 is a set of axioms that are consistent with the user's intuitions given the conversation between the two agents. We define the notion of the strongest set of theories that are consistent with the user's intuition to be the set of axioms from the repository from which the most can be proven. For example, if T_1 and T_2 are two theories from the repository, and T2 is a non-conservative extension of T_1, we say that T_2 is "stronger" than T_1. Lastly, to fully exploit the thrust of this algorithm, we capitalize on automated model generators such as Mace4 whenever we wish to construct a model. Figure 4-2 and Figure 4-3 below show a flow chart for the algorithm.

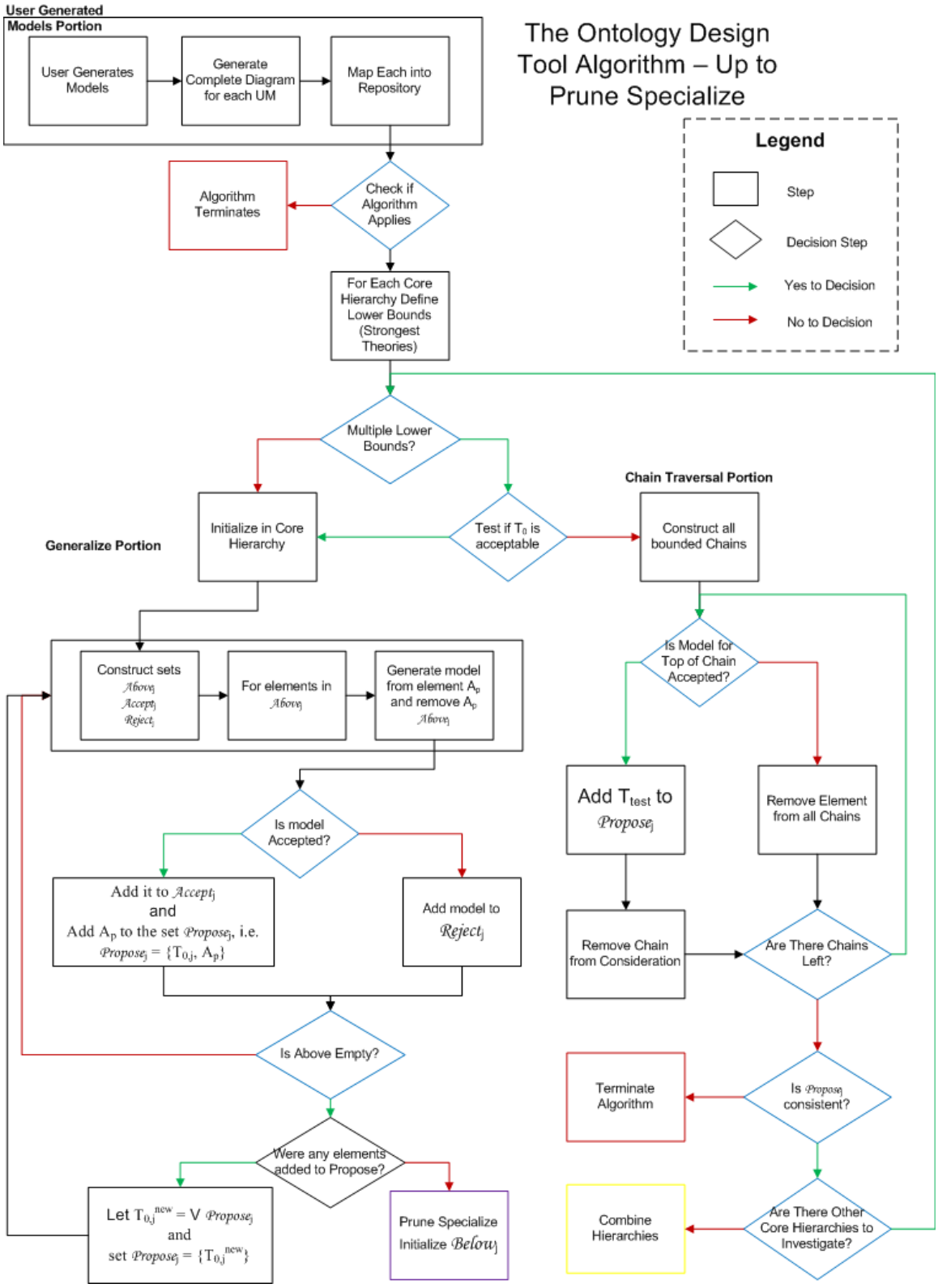


Figure 4-2 – Algorithm Flow Chart I

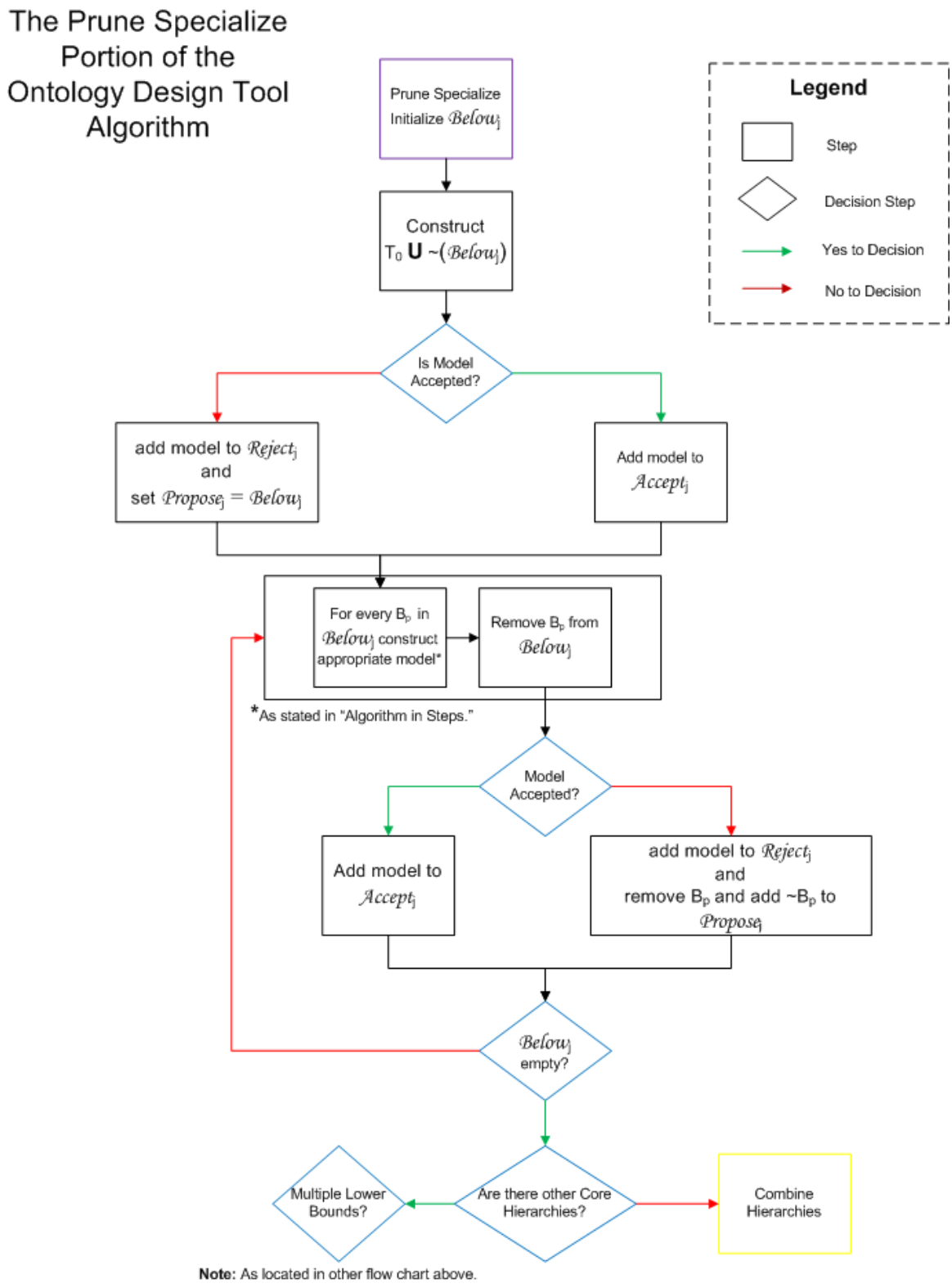


Figure 4-3 – Algorithm Flow Chart II

4.2.1 Ontology Design Tool Algorithm In Steps

In the following sections, the following notation will be used:

I is the users intended axioms

$CH_{m,n}$ is theory m located in hierarchy n in the ontology repository

$\mathcal{UM} = \{UM_i\}$ the set of user generated models

$\mathcal{SM} = \{SM_i\}$ the set of software generated models

---

Figure 4-4 – Algorithm in Steps

Inputs:

1. At least one user generated model that is interpretable by the ontology design tool.
2. Yes or no answers by the user to proposed models generated by the design tool

Outputs:

The set of strongest axioms with respect to the ontology repository that is consistent with the ontology designer's understanding as determined by the set of accepted and rejected models.

Steps:

1. **User Generated Models:** Using the Sandbox Tool, Agent A (the user) generates at least 1 model using their intuition.
2. Let $\mathcal{UM} = \{UM_i\}$ be the set of all models generated by the user.
3. **Generate Complete Diagrams:** For each UM_i, generate the complete diagram of the model

4. **Map to Core Hierarchies:** For every UM_i, test it for consistency against each $CH_{m,n} \in \mathcal{Hier}_j$,[3] using a breadth-first specialization algorithm for every core hierarchy.
    a. If $CH_{m,n}$ is consistent with UM_i, then $CH_{m,n} \in \mathcal{Consistent\_UM}_{i,n}$
5. **Check if Algorithm Should Terminate:**
    a. If for any $UM_{i,}$, every $\mathcal{Consistent\_UM}_{i,j}$ is empty, then display "The model you provided does not map into any theory in the repository, your concept utilizes axioms that cannot be represented via this algorithm." **End algorithm.**
    b. If for any j, the intersection of all $\mathcal{Consistent\_UM}_{i,j}$ is empty, then display "The models you provided map into different core-hierarchies, the algorithm cannot generate a set of axioms that will be consistent with your intuition." **End algorithm.**
6. **Initialize in each Core Hierarchy:** Let $\mathcal{Consistent}_j = \bigcap_i \mathcal{Consistent\_UM}_{i,j}$.[4]
7. **Rid Parents:** For every element $CH_{m,n} \in \mathcal{Consistent}_j$, if its child is also in $\mathcal{Consistent}_j$, remove $CH_{m,n}$
8. **Unique Lower Bound: Case 1** $\mathcal{Consistent}_j$ has 1 element
    a. Set $\mathcal{Investigate} = \{T_{0,j}\} = \mathcal{Consistent}_j$, GoTo **Define Sets**
9. **Multiple Lower Bounds: Case 2** $\mathcal{Consistent}_j$ has > 1 element
    a. Let $T_{0,j} = \bigvee_m CH_{m,j}$ where $CH_{m,j} \in \mathcal{Consistent}_j$, and let the set $\mathcal{Investigate} = \{T_{0,j}\}$[5]
    b. Let $\mathcal{Below}_j = \{B_p\}$.[6]
    c. **Test if $T_{0,j}$ is Acceptable:** Present a model for $T_{0,j} \cup \sim \mathcal{Below}_j$
        i. If model is accepted
            1. Initialize the set $\mathcal{Propose}_j = \{T_{0,j}\}$[7]

[3] The set of all theories in hierarchy j
[4] The notion of strongest theories for hierarchy j is selected over only elements in $\mathcal{Consistent\_UM}_{i,j}$.
[5] This set corresponds to all core-hierarchies which every UM_i mapped to and will thus be investigated.
[6] $\mathcal{Below}_j$ is the set of all theories connected to and immediately below $T_{0,j}$
[7] $\mathcal{Propose}_j$ is the set of axioms currently being proposed for the user's intuition

2. GoTo **Define Sets.**

ii. If model is rejected - **Chains Traversal:**

1. Let $\mathcal{LowerBound}_j = \mathcal{Consistent}_j$
2. Let $\mathcal{Propose}_j = \{\perp\}$
3. **Chain Construction** For every element in $\mathcal{LowerBound}_j$,
   a. create and store every possible chain to each connected $B_p \in \mathcal{Below}_j$ in $\mathcal{Possible\_Paths}_j$
4. **Traverse Chains:** For every chain in $\mathcal{Possible\_Paths}_j$
   a. Let T_{test} = top element of each chain
   b. Construct M_{Ttest} from $T_{test} \cup \sim\mathcal{Below}_{j,test}$
      i. if accepted add T_{test} to $\mathcal{Propose}_j$[8]
      ii. else, remove T_{test} from all chains contained in $\mathcal{Possible\_Paths}_j$, GoTo **Traverse Chains**
5. Let $H_{0,j} = \bigcup_i \mathcal{Propose}_j$
   a. if $H_{0,j}$ is consistent, GoTo **Check Complete Coverage**
   b. **Inconsistent Chain Union:** else display: "While the theories in this hierarchy of the repository are all individually consistent with the vetted models, they are not consistent with one another. The algorithm cannot determine the strongest set of theories for your relation/function." **End algorithm.**

10. **Repeat Generalize Test – Proposed** for each $T_{0,j}$
11. **Define Sets:**
    a. Initialize $\mathcal{Accept}_j = \mathcal{UM}$
    b. Initialize $\mathcal{Reject}_j = \{\perp\}$[9]
    c. Let $\mathcal{Above}_j = \{A_p\}$.[10]

[8] In this way, every theory that is consistent in the defined bound is preserved

[9] $\mathcal{Accept}_j$ is the set of models that are accepted by the user while $\mathcal{Reject}_j$ is the set of models that are rejected by the user

12. **Generalize Test:**
    a. For each element $A_p \in \mathcal{Above}_j$, construct a model $M_{Ap,j}$ for $A_p \cup \sim T_{0,j}$ and present it to the user[11]
    b. If $M_{Ap,j}$ is rejected by the user
        i. then add it to $\mathcal{Reject}_j$
        ii. remove A_p from the set $\mathcal{Above}_j$
    c. else
        i. add it to $\mathcal{Accept}_j$ [12]
        ii. add A_p to the set $\mathcal{Propose}_j$, i.e. $\mathcal{Propose}_j = \{T_{0,j}, A_p\}$
        iii. remove A_p from $\mathcal{Above}_j$
    d. When $\mathcal{Above}_j$ is empty
        i. if $\mathcal{Propose}_j = \{ T_{0,j} \}$
            1. go to **Prune Specialize**
        ii. else, $\mathcal{Propose}_j \neq \{ T_{0,j} \}$
            1. let $T_{0,j}^{new} = \bigvee \mathcal{Propose}_j$
            2. set $\mathcal{Propose}_j = \{T_{0,j}^{new}\}$
            3. goto **Generalize Test**

13. **Prune Specialize:**
    a. Construct model $M_{0,j}$ for $T_{0,j} \cup \sim \mathcal{Below}_j$
    b. **If:** $M_{0,j}$ is rejected then
        i. add it to $\mathcal{Reject}_j$
        ii. set $\mathcal{Propose}_j = \mathcal{Below}_j$
        iii. For each element B_p in $\mathcal{Below}_j$, construct a model $M_{Bp,j}$ for ($B_p \cup \sim \mathcal{Below}_j \setminus B_p$)
            1. If $M_{Bp,j}$ is accepted then add it to $\mathcal{Accept}_j$ and try the next element in $\mathcal{Below}_j$
            2. else add it to $\mathcal{Reject}_j$ remove B_p and add $\sim B_p$ to $\mathcal{Propose}_j$

---

[10] $\mathcal{Above}_j$ is the set of all elements in the hierarchy that are connected and directly above $T_{0,j}$
[11] As indicated earlier, an automated model generator may be used to construct a model for a particular set of axioms.
[12] This means $A_p \cap I \neq 0$, so our initial $T_{0,j}$ is too specific.

c. **Else:** $M_{0,j}$ is accepted add it to $\mathcal{Accept}_j$

i. For each element B_p in $\mathcal{Below}_j$, construct a model $M_{Bp,j}$ for B_p U ~ $\mathcal{Below}_j \setminus B_p$

1. If $M_{Bp,j}$ is accepted

a. add it to $\mathcal{Accept}_j$

b. try next element in $\mathcal{Below}_j$

2. if rejected

a. add it to $\mathcal{Reject}_j$

b. add ~B_p to $\mathcal{Propose}_j$

14. **Proposed Axioms from hierarchy j:** $H_{0,j} = \bigcup_i \mathcal{Propose}_j$.[13]

15. **Check for Full Coverage:** Check and see if there exists k ≠ j, where for any i, $\mathcal{Consistent\_UM}_{i,k}$ is non-empty

a. if every $\mathcal{Consistent\_UM}_{i,k}$ is empty, then display:

i. "The ontology design tool has determined that the set of axioms $H_{0,j}$ is collection of the strongest theories from the repository that are consistent with all the models you have accepted as being possible manifestations of your relation, and it is inconsistent with all those labeled as Reject." **End Algorithm**

b. if there exists a k ≠ j, where for every i, $\mathcal{Consistent\_UM}_{i,k}$ is not empty, then **End Algorithm** goto **Combine Hierarchy Algorithm.**

c. if there is at least one $\mathcal{Consistent\_UM}_{i,k}$ that is not empty, check every $CH_{m,k}$ against $H_{0,j}$ for consistency,

i. if all are consistent, display: "Some of the models you provided also mapped into another core-hierarchy. All those theories are consistent with $H_{0,j}$ but are inadequate to wholly describe your idea. Given the accepted and rejected models, the axioms of $H_{0,j}$ provide comprehensive cover for the relation in question." **End Algorithm**

[13] $H_{0,j}$ is the strongest set of theories for the user's intuition in core-hierarchy j.

ii. if there is at least one that is not consistent with $H_{0,j}$, then display: "Some of the models you provided also mapped into another core-hierarchy. Some of these theories are inconsistent with $H_{0,j}$. This means that there was some theory that applied to one of your provided models but not the others. Likely, that one theory was a match for a single example model, but does not account for the others. $H_{0,j}$ still provides comprehensive cover to the relation in question given the sets of accepted and rejected models." **End Algorithm.**

The process above allows us to reach a fairly detailed set of axioms that represent the categorized components of the ontology designer's intuition, *I*. If $I = T_i \cup O_i$, we remain silent about O_i, but can be sure that any T_i we provide is consistent with I. Before presenting the correctness proof however, it is necessary to touch on several points regarding the formulation of the algorithm, namely how information is learned by the system and how navigation is thusly propelled.

The core hierarchies mentioned above represent different containment hierarchies that can be constructed for various categories of mathematical structures, such as orderings, symmetries, groups etc. Each hierarchy is investigated as a silo. If theories in two hierarchies generate a non-conservatice extension, the hierarchies should actually be merged. If all user models mapped into more than one hierarchy, the results gleaned from the above process must now be tested against one another for consistency, since in general, only interactions among classes within each hierarchy are well understood and defined. Interactions between the different hierarchies require further investigation which is beyond the scope of this thesis.

4.2.2 Combine Algorithm In Steps

The following algorithm applies only when every user model mapped into at least two core-hierarchies simultaneously. This means that broadly, two "types" of theories were

consistent with every user provided model. The algorithm developed above would then create Reject and Accept sets for each hierarchy individually, leading to a bifurcation of tested models.

The end result is that the algorithm would develop $H_{0,1}$, $H_{0,2}$ etc. While each of these sets of theories is consistent with $\mathcal{UM}$, they are only consistent with their associated $\mathcal{Accepted}_j$ sets, not necessarily with models accepted from other core hierarchies.

In this situation there are number of scenarios. First, let's call the set of proposed axioms $\mathcal{Combine} = \{H_1, H_2, \ldots, H_n\}$. If all the elements of $\mathcal{Combine}$ are consistent with one another, then we can simply provide the union of all the axioms to the user, since the resultant theory is consistent with every $\mathcal{Accept}_j$ and inconsistent with every $\mathcal{Reject}_j$ by virtue of the extension of these theories being a conservative extension.

The situation becomes significantly more opaque if any two H_j's are inconsistent with one another. One possibility is that the repository has inadequate cover to be able to produce a satisfactory set of theories for the user. Another might be that perhaps the user is trying to define a relation or function over multiple types or sorts of arguments. In general, there is no way of knowing which case is applicable. Hence, the most that can be done in this scenario is to show all the consistency / inconsistency relations between every member of Combine. This scenario requires significantly more research and as such, if it is encountered, the Algorithm simply recognizes it and states that it is unable to develop a set of coherent axioms for the user's relation or function.

Figure 4-5 – Combine Hierarchies

Inputs:

Combine = $\{H_j\}$ which are the strongest theories from hierarchy j that are consistent with every UM_i and $\mathcal{Accept}_j$ but inconsistent with $Reject_j$. Consistency with $\mathcal{Accept}_k$, $k \neq j$ is unknown.

Outputs:

If every element of Combine is consistent with one another:

- H_0 = the set of the strongest theories from the repository that are consistent with every UM_i and every model in *Accept* and inconsistent with every model in *Reject*.

1. **Combine Hierarchy Algorithm:** Store every $H_{0,j}$ in the set Combine
    a. If Combine is a consistent set of theories, then display:
        i. "The strongest set of axioms from the repository which correspond to your intuition are $\cup H_{0,j}$" **End Algorithm.**
    d. Else present the following to the user:

"The ontology design process has determined that your intuition maps disjointly into several theories in the hierarchy. Interaction between these theories is unknown. In general, this indicates one of three things:

1. The repository is not expressive enough to capture the intended axioms for your relation
2. You may be trying to define a relation that behaves different depending the sorts or types of arguments.
3. The axioms you desire is a disjoint union of the theories listed above.

The design tool can not distinguish these cases and is unable to propose an adequate set of axioms for your relation/function."

4.2.3 Part 1 – Eliciting User Models

An ontology designer has an internal idea which is expressible in FOL but lacks the necessary background to articulate axioms for the idea. Simultaneously, there exists a repository of catalogued and axiomatized logical structures from which we try to define the user's intuition. These structures may be organized into a set of containment

hierarchies, corresponding to $CH_{m,n}$. Given that we wish to avoid using FOL syntax to communicate, we look for an alternative tool.

We note that any set of axioms, *I* has associated with it the set M^I, corresponding to its models. The algorithm begins by working backwards; so long as the user can generate at least one model, a conversation may be initiated between the user and the ontology design tool. An ontology designer is first asked to "draw" or generate at model using one of the supported representation tools (i.e. graphs). Each user generated model, UM_i is then translated into sets of FOL statements and tested for consistency against each theory $CH_{m,n}$, in the repository. By eliciting a user-generated model, we are trying to locate which theories in the repository are likely candidates for a match for the user's intuition. However this also draws attention to some of the ambiguities introduced when considering only particular models and not the underlying theories that lead to their construction.

For the first part of the algorithm, this is not as important since we are only trying to find the strongest theories that are consistent with the user generated models. It is when the system begins providing models to the user that this issue becomes more important. To situate the designer within the repository, we implement a breadth-first specialization search through each core-hierarchy separately. A core hierarchy is defined as one where there is only one root for all the classes beneath and whose theories share no non-conservative extension with those of another hierarchy. Any further interaction between the hierarchies is left undefined.

We may employ a breadth-first specialization algorithm since extensions of theories correspond to moving down a core hierarchy. Namely, as more axioms are added, more is entailed by resultant theory; these additional axioms define an increasingly restricted model space, since they curb the number of models which are acceptable. It is also in this sense that we define a concept of "stronger" set of axioms. A set of axioms is said to be "stronger" if one can prove more things about the world than another set of axioms.

For each core-hierarchy we can construct a set, $\mathcal{Consistent\_UM}_{i,j} = \{CH_{m,j}\}$, that is consistent with a particular user generated model. We then repeat this for each supplied user model. However, before we move to the next stage of the algorithm, we must first acknowledge a limitation of an approach relying on predefined logical constructs.

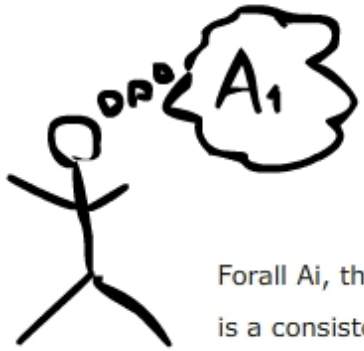
Consider when a scenario where an ontology designer inputs two models UM_1 and UM_2 that map only into CH_1 and CH_2, two different core-hierarchies respectively. This means that the models which represent the user's idea are not captured wholly by any existing class of axioms. Namely, UM_1 is *not* consistent with any theory in CH_2 and symmetrically UM_2 is not consistent with the CH_1. If this scenario arises, the algorithm developed here does not apply and the ontology design tool cannot help the user derive appropriate axioms. There are variations on this situation which the algorithm *can* address, but that will be left for the correctness proof. We define a necessary condition for the algorithm to work here: it suffices to say that given a set of user models $\mathcal{UM}$ and a core-hierarchy; there must be at least one theory in some core hierarchy that is consistent with every user-generated model.

This restriction does not mean that every UM_i must be consistent with only one core-hierarchy. Only that there exists at least one core-hierarchy in which every user model maps into. If the above condition is satisfied, then the algorithm applies and we move to the next stage considering each appropriately "covered" CH.

We now continue exploring each CH in isolation. At this point, if there is only one element in $\mathcal{Consistent}_j$, we set $T_{0,j}$ to that element. This situates the user somewhere in $CH_{m,j}$ and concluding the first component of the algorithm. The initiative in the conversation is now passed to the ontology design tool.

If there is more than one element in $\mathcal{Consistent}_j$, we let $T_{0,j}$ be the global union of $\mathcal{Consistent}_j$. We construct a model for $T_{0,j}$ and if the user deems it acceptable, we conclude the first part of the algorithm as above. If however, that model is rejected, we must instead investigate every chain between $T_{0,j}$ and the element of $\mathcal{Consistent}_j$. The conclusion

of this investigation might yield a set of acceptable theories that mutually are consistent (hence generated the strongest theories from hierarchy j consistent with the user's ideas) Otherwise, the chains are inconsistent and the algorithm is faced with the situation of being unable to disambiguate the appropriate case. Thus it simply lists these cases and terminates.

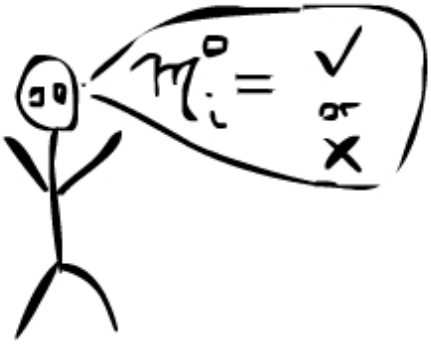


A1 = Ti U Oi, where Ti is a class of axioms belonging to the set of mathematical structures, Oi are sets of axioms that are uncategorized

Forall Ai, there exists the set M(a), where each M(a)i is a consistent model for Ai.

However, Agent A, might not be comfortable articulating her intuition in the language of FOL. Therefore, communicate using models (M(a)i) instead of axioms, Ti.

User instead constructs >0 models for their intuition A. These models numerate the set M(a)i. Each M(a)i is then individually mapped to the most specific Tj with which it is consistent.



Let Am be the set {Tj} where each Tj is one which was consistent with a user generated M(a)i. Let T(0) be the global join of Am. Now Agent B constructs models based on T(0) and shows them to agent A. They are rejected or accepted and navigation of hierarchy occurs.

Figure 4-6 – Algorithm Overview

4.2.4 Part 2 – System Generated Models and User Feedback

The following process further expands on the interaction between models and sets of axioms. Based on $T_{0,j}$ and its immediate neighbourhood in the containment hierarchy, the design tool must now construct models to provide to the user and guide itself to the most appropriate set of axioms. Using this information, we shall ask questions that will efficiently minimize the ambiguity that results from communicating via models.

One such ambiguity is that we cannot guarantee that any particular presented model doesn't correspond to a "special" case. For example, if one is tasked to provide an example triangle, many will inadvertently draw an isosceles triangle. Now if one were to subsequently begin inferring properties about triangles based solely on this one model, it might very easily lead to the addition of accidental (and unwanted) restrictions to the notion triangle, such as "all triangles have two sides of the same length." Clearly an isosceles triangle is not a *prototypical* triangle.

Since we wish to bypass the syntax and grammar of FOL and instead communicate solely on the semantics, we should consider what communication is being exchanged. The conversation is fairly simple here it, is like a pre-language being pointing to things that exist in the world and asking another whether that corresponds to what they are thinking. We provide the user with a possible model of their intuition and ask them if the model is acceptable or not.

As a result, in constructing the algorithm we must be very careful in terms of what the acceptance and negation of models imply for our understanding of an ontology designer's intuitions. Two cases arise corresponding to what a yes or no means, which are outlined below:

Case 1: M^0 is accepted
This only tells us that that $T_0 \cap I \neq 0$.

that is to say, we have generated a model based on T_0 which is also in the user's intended model space. So the intersection between models generated by T_0 and I is not empty. This isn't a very strong result, yet it still drives one direction of the algorithm, depending on context of the algorithm as described below. Briefly, this result may be used to push the algorithm "upwards" or to more general A_p's.

Case 2: M^0 is rejected
Here, we derive a stronger result, namely that:

1. M^0 does not belong to M^I

which implies that

2. I is not contained in T_0

If T_0 allows us to construct a model which is not in the user's model space, then it is missing important axioms. This can be used to push the algorithm "downwards," that is to a stronger, more specialized theory.

Knowing what an "Accept" or "Reject" from a user means for the theory T_0 which generated the candidate model, influences the developed algorithm. In many ways, the algorithm developed here is the application of the scientific method of inquiry. Since it is impossible to enumerate all the possible models of any theory to the user, we can never conclusively prove that T_0 is the desired set of axioms. All we can hope to do is propose models to test which if rejected would falsify T_0 as the appropriate theory. Each construction of M^0 is in essence an experiment aiming to falsify the claim that T_0 is contained in I.

With this insight, let us consider the neighbourhood in which the $T_{0,j}$ as determined by part 1 is located. Based on the user's input, we have situated ourselves at $T_{0,j}$, which in general, is located in some containment hierarchy as below in Figure 4-7.

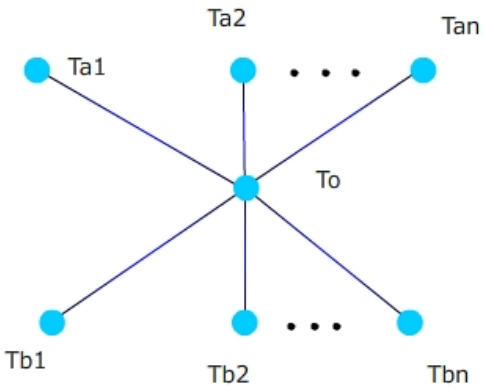


Figure 4-7 – $T_{0,j}$ in Hierarchy

Each T_{ai} are classes of axioms that T_0 is an extension of, while each T_{bi} in turn represents an extension of T_0. Thus T_{ai} are more general than T_0, and T_{bi} are more specific.

If $|Consistent_j| > 1$, then T_0 might not necessarily be a set of desired axioms, since it may include models that are too general for the user. Hence, we must test to see if the models of T_0 are acceptable, which we achieve by constructing a model of $T_{0,j} \cup \sim Below_j$. If this model is Accepted, we proceed as described two paragraphs below. If it is rejected, the algorithm instead conducts the subchain algorithm. Here, we know that the strongest theories in hierarchy j in the repository is bounded by $Consistent_j$ and $T_{0,j}$. Since $T_{0,j}$ was rejected, we must construct a chain for each element in $Consistent_j$ to $T_{0,j}$. For each chain, we test a model from the topmost element, and if it is accepted, we add it to $Propose_j$, otherwise we remove the topmost and continue down the chain until in the limit, we reach the elements of $Consistent_j$ which we already know are consistent with the user's intuition.

Once every chain has been traversed, and the most general element of each chain is stored in $Propose_j$, we must now see if the union of these elements is consistent. If so, we have constructed the strongest theories from hierarchy j for the user's relation/function. If not, there are a multitude of possible cases which the algorithm cannot distinguish between, and hence terminates.

If $|Consisten_j| = 1$ or the initial T_0 was Accepted, we must accommodate the possibility that the models the user provided was accidentally too specific (i.e. an "isosceles triangle"). Hence, we generate software models to test whether weaker theories are acceptable. This means constructing models from each T_{ai}. If a model from one of these theories is accepted, it implies that T_0 was too specific, and we need to generalize. Consequently, we would move "up" in the hierarchy, and set our new T_0 to T_{ai}. As noted above, there is an additional piece of nuance here in constructing the model for T_{ai}. Since T_0 is an extension of T_{ai} we cannot simply construct a model from T_{ai}. Instead we must explicitly deny those models that are consistent with T_0, i.e. ensure that we get at least a scalene triangle of some sort. This can be achieved by generating a model from $T_{ai} \cup \sim T_0$.

If we consider what is implied when a user rejects these models, we know that T_{ai} represents a theory that is too general. Unwanted models are admissible. In such a way, all the parents of any T_0 may be pruned, or else the T_0 goes "up" until it hits the root. When it is no longer possible for the algorithm to move up (i.e. all parents have been pruned), attention is then focused on what can be said about the children of T_0.

The situation here is a bit trickier, and it is useful to note that each T_{bi} corresponds to a set of nested sets of models, much like Russian dolls, as illustrated in Figure 4-8 below.

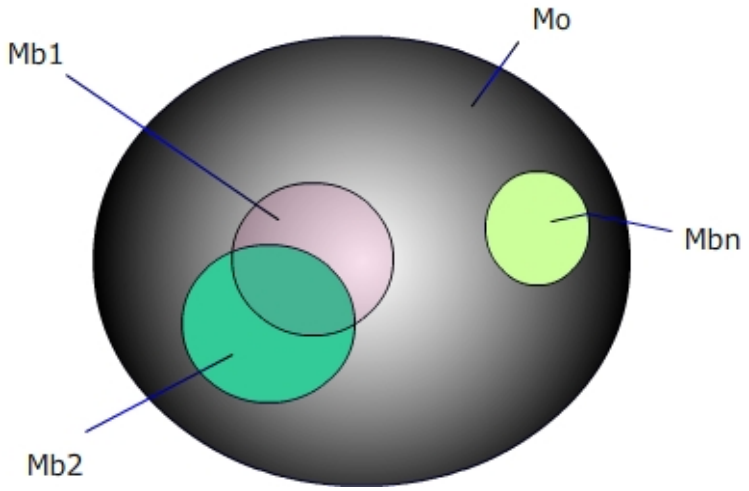


Figure 4-8 – Models of T_0

This presents a "down map" of meaningful statements about the world that the algorithm shall now attempt to traverse. It should be noted that here we are using the term meaningful in the context of knowledge represented in a repository. The repository is a collection of ways to understand the world, often captured through models.

When the algorithm can no longer generalize, it then tries to refine its search. This corresponds to the *Prune Specialize* steps above. Since for any T_0, the model space of the union of all T_{bj} is contained T_0. To refine the user's intuition, we then must investigate models that are particular to localized areas in this map of M_0.

This also means the generation of models to investigate meaningful areas of M_0, which result in the construction "artificial classes" of axioms. These classes are combinations of pre-existing theories that have not yet been named. It is in this sense that they meaningfully divide the set of models for T_0, M_0.

We also know that in general, the children of T_0 do not necessarily provide a full partition of M_0 only that they are contained in it. Each element of $\mathcal{B}elow_j$ already represents a meaningful subspace of M_0, but what of the "negative" spaces, illustrated by the uncovered (grey) areas in Figure 4-8? These are what the artificial classes alluded to above will attempt to cover. We recall that the acceptance of a model can only show that those below are too strict, while rejection allows us to "chop off" a model space (and its corresponding theory). If we learn that the user specifically does not want an "isosceles triangle," then the axioms we want are those for triangles that do not have two sides equal.

The number of meaningful spaces in the provided map that can be constructed based on one level of depth in an explicit repository is as follows:

Given:

M_0 representing the entire model space

$\mathcal{Below}_j = \{B_p\}$, that is the powerset of all the named children of $T_{0,j}$ (in the following we drop the second index since we know we are in core-hierarchy j.)

Meaningful spaces:

1. T_0 which is all of M_0
2. $T_0 \cup \sim\mathcal{Below}_j$ – M_0 excluding all of its children
3. $\cup\ \mathcal{Below}_j$ – only the area of M_0 covered by the children of T_0
4. $\mathcal{Below}_{j\ i} \cup \sim (\mathcal{Below}_j \setminus \mathcal{Below}_{j\ i})$

The first is trivially T_0 itself. The second corresponds to the negative spaces in Figure 4-8 above, assuring that models from T_0 excluding its possible extensions are explored. The third construct examines combinations of the children. Finally, the last combines 2-3, thus providing a complete cover of the model space.

Unfortunately the resultant search space can become quite large and grows exponentially. We want to learn the most information from each test, so we can use this to inform how to order the search. Thus we would like to order the elements according to the strongest theories we can construct, allowing us to lop off (since we can only falsify) the largest and least ambiguous piece as possible with each test.

The largest, least ambiguous piece would be $T_0 \cup \sim B_1 \cup \ldots \cup \sim B_n$. One could test $T_0 \cup \sim B_p$ individually, but this is an ambiguous model space, since it encompasses only the negation of B_p. While this theory *might* be what the designer wishes, we will derive it through different means.

If the resultant model from negation of children theory is rejected, then we immediately know that we are interested in constructs 3 and 4 as defined above. If it is accepted, we gain little information and must carve the meaningful spaces into smaller, more precise

components. In this portion of the algorithm, whenever $M_{(test)}$ is rejected we must refine the test theory to reflect the rejection:

$\mathcal{Propose}_j$ = unchanged if model accepted
$\mathcal{Propose}_j$ = revised if model rejected

The revision requires some thought to clarify what is done depending on the model being tested. We recall that we can only falsify theories, not positively prove them. With this in mind, we can define two cases:

1. $T_{(test)}$ consists of T_0 and $\sim B_p$
2. $T_{(test)}$ is only composed of combination of $\mathcal{Below}_j = \{B_p\}$

What we are falsifying in 1 is simply the negation of all the children. We are testing to see if T_0 is desired to be positively asserted or not. If it is, we grow our proposed axioms by incrementally adding each B_p that is rejected, as determined by the $T_{(test)}$'s constructed from 2.

If $M_{(test)}$ is the model generated from $T_0 \cup \sim B_1 \cup \ldots \cup \sim B_n$ and it is rejected, then $\mathcal{Propose}_j = \mathcal{Below}_j$. Moreover, it means that the space of the complement of the children is not in the user's intended theory and the desired axioms are some union of B_p. Now, instead of having $\mathcal{Propose}_j = T_0$ and simply adding negations of its children, we begin with the strongest theory and change any positive to negative if falsified. The revision function in this case would alter the proposed axioms thusly:

If $M_{(test)} = B_p \cup \sim \mathcal{Below}_j \setminus B_p$ then
$\mathcal{Propose}_j = B_1 \cup \ldots \cup B_{p-1} \cup \sim B_p \cup \ldots \cup B_n$

This revision is reflected in Case 2 of the Prune Specialize section of the algorithm above. In such a manner, we ensure that given the theories available in the repository, we

construct the strongest set of theories, i.e. those with which we can entail the most, that are consistent with models the user accepts and inconsistent with those it rejects.

Finally, if the scenario arose where the all the user's models mapped into more than one core-hierarchy the algorithm would test for consistency across these "silos." If all of the individual strongest theories were consistent with one another, then the ontology design tool has found the strongest axioms in the repository that satisfy the user's understanding of their idea as determined by interaction with the tool. We recall that the combination of silo's yields only conservative extensions of each silo.

If the various strongest theories are not consistent with one another, the algorithm has reached a limitation of the ontology and perhaps axioms that are not represented in the hierarchy are required to fully capture the user's idea. There is a possibility that the user is unintentionally defining a relation or function over multiple types of sorts. However, given the level of ambiguity, it cannot provide a conclusive result for the user's idea given the interaction.

To recap, the Figure 4-9 below is a simplified schematic of the algorithm:

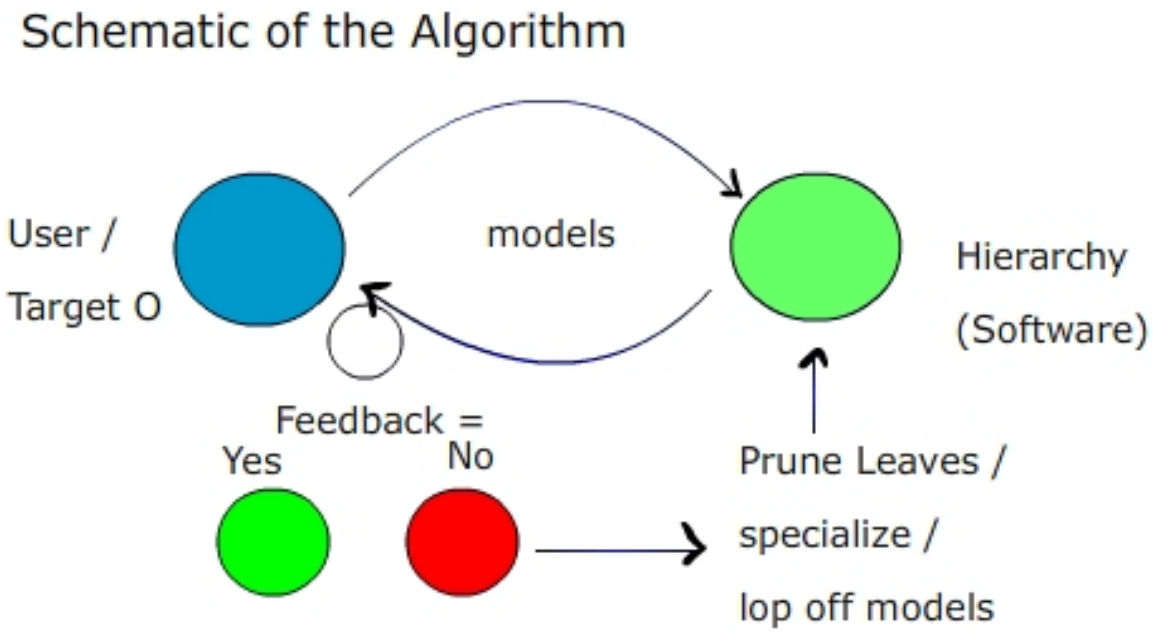


Figure 4-9 – Algorithm Schematic

The process illustrated above requires the following components:

- an ontology designer (Agent A) who can:
  - make at least 1 model of the world using their relation
  - can recognize presented models as being acceptable or not
- a software agent (Agent B) who can:
  - navigate a containment hierarchy of logical structures
  - generate models based on the axioms in the hierarchy
- an interface to interpret Agent A's models
- an interface to communicate models from Agent B to A and get feedback

4.2.5 Correctness Proof

In the steps section of the algorithm above, we needed to check if the algorithm was capable of assisting the ontology designer. Specifically, it is necessary to determine if the user generated models map to the repository in such a way that the criterion of falsifiability may be applied. To do this we simply require that there exist at least one core-hierarchy which every user generated model is consistent with. Lemma 4-1 captures this requirement.

Lemma 4-1 (Applicability of Algorithm)

The set $\mathcal{UM}$ satisfies the following property

- *The root of at least one core-hierarchy ($CH_{1,n}$) is consistent with every element of $\mathcal{UM}$*

Proof. There are four cases that we need to consider that are all captured when there are at least two user models, UM_1 and UM_2.

Case 1:

UM_1 does not entail any CH1,n

UM_2 irrelevant

In this case, the step **Check if Algorithm** would see that $\mathcal{Consistent\_}\ \mathcal{UM}_{i,j}$ is empty for UM_1 (**Step 5a**) and would end the algorithm.

Case 2:

Assume that UM_1 is only consistent with $CM_{i,1}$ and UM_2 is only consistent with $CM_{k,2}$. By virtue of the breadth first specialization algorithm this means that UM_1 and UM_2 are consistent with roots of each hierarchy, $CM_{1,1}$ and $CM_{1,2}$ respectively. However, by the same algorithm it means that UM_1 is inconsistent with $CM_{1,2}$ and UM_2 is inconsistent with UM_1. This means that there exist no theories in the repository that are consistent with all of the user's intended models.

UM_1 entails $CH_{1,1}$	UM_1 does not entail $CH_{1,2}$
UM_2 does not entail $CH_{1,1}$	UM_2 entails $CH_{1,2}$

In this scenario the ontology design tool cannot be used to generate the strongest set of theories since there are no theories in the repository that are consistent with the user's intuitions as expressed through their models. This is captured in the algorithm also by the step **Check if Algorithm Applies** corresponding to **Step 5b** resulting in termination of the algorithm and the corresponding message as above.

Case 3:

UM_1 entails $CH_{1,1}$	UM_1 does not entail $CH_{1,2}$
UM_2 entails $CH_{1,1}$	UM_2 entails $CH_{1,2}$

As long as there is at least one hierarchy in which there is some theory that is consistent with all the user models then a theory may be found. In this case, the algorithm would be able to propose axioms from core-hierarchy 1 since it was consistent with all supplied user models. In this case, each $\mathcal{Consistent}\_\ \mathcal{UM}_{i,j}$ is non-empty and there is at least one core-hierarchy which is consistent with both user models. **Step 6** captures this scenario by taking the intersection of each $\mathcal{Consistent}\_\ \mathcal{UM}_{i,j}$ and storing it in $\mathcal{Consistent}_j$ as required above.

Case 4:

UM_1 entails $CH_{1,1}$	UM_1 entails $CH_{1,2}$
UM_2 entails $CH_{1,1}$	UM_2 entails $CH_{1,2}$

The last case occurs when all user generated models map into more than one hierarchy. Again Step 6 would generate at least one $\mathcal{Consistent}_j$ which satisfies the property listed above.

Consequently, this Lemma is satisfied since Cases 1 and 2 exit the algorithm, while 3 and 4 satisfy the property for UM as desired. **QED.**

In general, when Lemma 4-1 is satisfied by more than one core-hierarchy we do not know if the Ontology Design Tool algorithm applies. There is one case where the outputs can be guaranteed, as shown in Lemma 4-4 below, otherwise we can only acknowledge the limitations of the ODT algorithm and output the process.

Lemma 4-2 (Map into Hierarchy)

For a set of user generated models, $\mathcal{UM}$, one may construct the set Investigate = $\{T_{0,j}\}$ which exhibits following property:

- *Each $T_{0,j}$ is the strongest theory from core-hierarchy j that is consistent with $\mathcal{UM}$.*

Remark. We recall that each hierarchy is a join-semi lattice, with each lower theory representing a non-conservative extension of those above. Moreover, since we are testing for consistency via only a subset of models, the provided models may be consistent with distinct "strongest" theories. Additionally, the intersection of set of consistent theories for each user generated model may also be more than one theory. If it is only one theory, we know it is the strongest. If the intersection yields more than one theory, a different algorithm is needed.

Proof. By Lemma 4-1 we know that there exists at least one core-hierarchy, j, such that every UM_i is consistent at least with its root. The breadth-first specialization search in the same step stores $CH_{m,n}$ in $\mathcal{Consistent\_UM}_{i,j}$ only if the complete diagram for that model is consistent with $CH_{m,n}$. The specialization search corresponds to traversing the theories in repository j from weakest to strongest. Moreover, since only consistent theories are stored in $\mathcal{Consistent\_UM}_{i,j}$, every theory up to the one which is no longer consistent with $\mathcal{UM}$ is stored.

For a fixed j, let $\mathcal{Consistent}_j$ be the intersection of every $\mathcal{Consistent\_UM}_{i,j}$. Step **Rid Parents** above, ensure that for every $CH_{m,n}$ in $\mathcal{Consistent}_j$ if it has a child, $CH_{m,n}$ is removed from $\mathcal{Consistent}_j$. At this point, there are two cases:

Case 1

There is only one element in $\mathcal{Consistent}_j$.

The intersection of each $\mathcal{Consistent\_UM}_{i,j}$, followed by pruning all the parents yielded a unique element. Since each $\mathcal{Consistent\_UM}_{i,j}$ contained every theory that was consistent with user model i, their intersections yields those theories that are consistent with all $\mathcal{UM}$. Pruning the parents would yield the strongest theory, since if there were any stronger theory, it would be captured in the intersection of $\mathcal{Consistent\_UM}_{i,j}$.

Case 2

There is more than one element in $Consistent_j$.

In this case, we let $T_{0,j}$ be the join of the resultant elements in $Consistent_j$. At this point, we know that the strongest theory in the repository, if it exists, is bounded at the top by the root of the hierarchy, and at the bottom by the union of the elements of $Consistent_j$. Moreover, if the model generated by $T_{0,j} \cup Below_p$ is accepted, then it is the new lower bound for the strongest theory in hierarchy j. If it is rejected, the Chain Traversal algorithm applies, and Lemma 4-5 proves that one can still construct the strongest theories in hierarchy j for the user's relation/function.

Lastly, the above holds for each hierarchy individually. Consequently, the set $Investigate = \bigcup_j T_{0,j}$ consists of each of the strongest theories in each hierarchy, with the properties as desired. If there is only one element in *Investigate*, Case 3 in Lemma 4-1 holds, otherwise we will need to invoke Lemma 4-4 as well. **QED.**

Let SWM correspond to the set of all software generated models as constructed in steps **Generalize Test** and **Prune Specialize** in the algorithm above.

Lemma 4-3 (Navigate Hierarchy)

Given a theory $T_{0,j}$ in core-hierarchy j, and SWM, then $H_{0,j}$ is the set of axioms which exhibits the following property:

1. *are the set of the strongest theories in core-hierarchy j*
2. *are consistent with every element of $Accept_j$*
3. *are inconsistent with every element of $Reject_j$.*

Remark. This component of the algorithm uses each core-hierarchy as a map through which to generate models to show the ontology designer. The hierarchy determines which models will be generated, and user feedback separates each model into the $Accept_j$ or

$\mathcal{Reject}_j$ sets. Since the most information is gleaned when an "experiment" model is falsified (rejected), intelligently navigating the model space is important.

The first step is to make sure that the current selected theory is not accidentally the strongest. Thus we need to test whether a model generated from a guaranteed more general theory is acceptable to the user. Consequently, for each parent theory, we construct a model asserting the parent and negating the child (thus ensuring that the resultant model isn't a special case again). Step **Generalize Test** conducts a depth-first generalization test, ensuring that models which would confirm that $T_{0,j}$ is too specific are presented to the user. Once the parent space is exhausted, one may now try to derive the strongest theories in the hierarchy by looking at the children space of the selected theory.

As per the discussion above, the question now arises as to how we may define meaningful unambiguous sub spaces. Given theory $T_{0,j}$, the largest unambiguous model space is defined $T_{0,j} \cup \sim\mathcal{Below}_j$, where the second term is the negation of all the first term's children. The **Prune Specialize** steps of the algorithm systematically go through each meaningful subspace in the model space for $T_{0,j}$. Again since we only really gain information when a proposed statement is negated, we can increase the "strength" of $\mathcal{Propose}_j$ by adding negations of theories in the repository. Once no more interesting models may be constructed, the union of the elements of $\mathcal{Propose}_j$ result in the strongest theories in hierarchy j that are consistent with all of the software generated models that the user denoted as "Accept" and inconsistent with all those denoted "Reject" and stored in their corresponding sets. Since the ontology designer judged each model according to whether it represented a possible configuration of their intuition, $\mathcal{Propose}_j$ is consistent with the user's intuition.

Proof. We recall that $\mathcal{A}ccept_j = \mathcal{UM} \cup \{\mathcal{SWM}_j\}$, where $\mathcal{UM}$ is the set of user models, and $\mathcal{SWM}_j$ are those models generated by the ontology design tool that the user has marked as "Accept."[14]

The steps for **Generalize Test** ensure that the axioms in $\mathcal{P}ropose_j$ satisfy conditions 2 and 3 above simultaneously. A more general (weaker) theory is required if a model corresponding to it is accepted, which corresponds to **Step 11-c**. If models for a more general theory are rejected, the axioms in $\mathcal{P}ropose_j$ are still inconsistent with $\mathcal{R}eject_j$.

In addition, the **Prune Specialize** steps satisfy conditions 2 and 3 by virtue of adding or removing axioms to $\mathcal{P}ropose_j$ in accordance with the acceptance or rejection of models. Bearing in mind that the notion of strongest has been defined relative to each core-hierarchy, we note that by **Prune Specialize** also ensures that the elements of $\mathcal{P}ropose_j$ are the strongest theories in the repository.

Finally, $H_{0,j}$ is simply the union of all the axioms in $\mathcal{P}ropose_j$ which as we have shown above are consistent with every element of $\mathcal{A}ccept_j$ and inconsistent with $\mathcal{R}eject_j$. Thus H exhibits properties 1-3 as desired. **QED.**

If as in Lemma 4-1, Case 4 applies then there are n core-hierarchies to investigate. Lemma 4-3 above shows that for each hierarchy we may determine $H_{0,j}$ as the strongest theories from it that are consistent with the user's intuitions as determined by labeling software models "Accept" or "Reject." We can define the set $\mathcal{C}ombine = \{H_j\}$ and provide the user with the strongest theories which are consistent with the user's intuitions as defined by the acceptance or rejection of models, otherwise provide the user with acknowledgement of limitations of ODT algorithm, and a list of possible scenarios.

Lemma 4-4 (Combining Hierarchies)

The set $\mathcal{C}ombine = \{H_i\}$ has the following properties:

[14] The marking of "Accept" means that these models are consistent with the user's intended models, further implying that the underlying theories from the repository that generated the models are consistent with the user's intuition.

1. *are the strongest theories from the repository that are:*
2. *consistent with every user generated model* UM_i
3. *consistent every accepted model* $\epsilon \mathcal{Accept} = \bigcup_j \mathcal{Accept}_j$
4. *inconsistent with every rejected model* $\epsilon\ \mathcal{Reject} = \bigcup_j \mathcal{Reject}_j$

Remark. We recall that each core-hierarchy is defined such that there does not exist theory T, such that T is a non-conservative extension of theories N and M from core-hierarchies i and j. If such a theory exists, the i and j ought be one single core-hierarchy.

Proof. This scenario arises when the set of user models, $\mathcal{UM}$ mapped completely into more than one core-hierarchy. Since the algorithm considers and generates models for the user to vet for each hierarchy independently, it is now necessary to see if it is possible to unify the results.

There are two ensuing cases:

Case 1

$\bigcup_j H_{0,j}$ is consistent (model constructed)

Step 1a of the **Combine Hierarchies** algorithm above tests this case. Since the union of these axioms can only be a conservative extension, we immediately know that the resultant model space is the same as the union of the sets of model for each of the individual theories. By definitions, it is the strongest set of theories from the repository since it is also consequently consistent with every software generated model labeled "Accept" and inconsistent with those labeled "Reject." Thus it is consistent with every element of $\mathcal{Accept}$ and inconsistent with every element of $\mathcal{Reject}$. We have thus satisfied the properties listed above.

Case 2:

There exists at least one $H_{0,j}$ that is inconsistent with the other elements in $\mathcal{Combine}$.

In this case, the models desired by the user do not intersect. Knowing that such an inconsistency exists means that the algorithm is aware of its limitation. Moreover, this means that there is at least one model space that is disjoint from all the others. This scenario is captured by **Step 1b** of the **Combing Hierarchies** algorithm resulting in simply displaying associated message as above.

Cases 1 and 2 cover the possibilities that may arise, with the latter exiting the algorithm and the former satisfying all the properties of this lemma as desired. **QED.**

Lemma 4-5 (Chain Traversal)

A set theories $H_{0,j}$ may be generated by the tool that satisfies the following properties:

1. *$H_{0,j}$ is the set of the strongest theories in hierarchy*
2. *$H_{0,j}$ is consistent with all user generated models.*
3. *$H_{0,j}$ is consistent with all software generated models that the user denoted "Accept"*
4. *$H_{0,j}$ is inconsistent with all software generated models that the user denoted "Reject"*

Proof. There are two cases here. Either the elements of $H_{0,j}$ are consistent, or they are not.

Case 1:

$H_{0,j}$ is consistent.

By Lemma 4-1 we know that the elements of $Consistent_j$ are the strongest possible theories in hierarchy j that are consistent with the user models. Moreover, Lemma 4-1 showed that it is consistent with all the user models, satisfying property 2. If a model for $T_{0,j}$ is rejected, we then know that the strongest theories are bounded by $T_{0,j}$ and the elements of $Consistent_j$.

We investigate each chain, checking to see if the topmost element is acceptable to the user. If it is, we know that it is the strongest theory in that chain. Repeating this process for each chain, we eliminate those theories that are too general, since their models are rejected by the user, satisfying property 4. Each model that is accepted, has its corresponding theory added to $\mathcal{P}ropose_j$ the union of which is $H_{0,j}$, satisfying property 3.

It remains to show that $H_{0,j}$ is the strongest set of theories. If for any theory T in $H_{0,j}$, there were a stronger theory T', that would mean that one of the software generated models marked as Accept by the user was inconsistent. This results in a contradiction, hence only the strongest theories that are acceptable to the user are in $H_{0,j}$, satisfying property 1.

Case 2:

$H_{0,j}$ is inconsistent.

Here, while individual theories in the repository are consistent with all user generated models and those models marked as acceptable, they are inconsistent with one another. There are a number of possibilities which the algorithm cannot distinguish between, hence Step **Inconsistent Chain Union** terminates the algorithm.

QED.

Theorem 4-1 (Ontology Design Tool)

A set of the strongest theories $H_{0,j}$ may be generated by the tool that satisfies the following properties:

1. *$H_{0,j}$ is a composite of the theories from the ontology repository*
2. *$H_{0,j}$ is consistent with all user generated models.*
3. *$H_{0,j}$ is consistent with all software generated models that the user denoted "Accept"*

4. *$H_{0,j}$ is inconsistent with all software generated models that the user denoted "Reject"*

Proof. By Lemma 4-1 we showed that the algorithm will engage only if it is possible to construct a theory that is consistent with every user generate model.

Through Lemma 4-2 we showed that we can determine the *strongest* theories in the core-hierarchy that are consistent with all user generated models.

In Lemma 4-3 we showed that we can also derive the strongest theories in the repository that are consistent with software generated models the user marked as "Accept" and inconsistent with those marked as "Reject" satisfying properties 3 and 4 for one hierarchy.

Lemma 4-4 shows that the algorithm either allows each of the $H_{0,j}$'s to be combined together if they are mutually consistent, otherwise that it exits.

Finally, since the starting theory for Lemma 4-3 is $T_{0,j}$ with which satisfies the properties of Lemma 4-2, we ensure that the output of the algorithm $H_{0,j}$ satisfies all four properties listed above simultaneously. We recall that Lemma 4-2 stipulated that $T_{0,j}$ be the strongest theory in a theory that was consistent with $\mathcal{UM}$ while Lemma 4-3 ensured that $H_{0,j}$ be the strongest that were consistent with the software generated models as collected in $\mathcal{A}ccept_j$ and inconsistent with the software generated models are collected in $\mathcal{R}eject_j$. Combining these two results satisfies all four properties above as required. **QED.**

4.2.6 Extending the Algorithm

To recap, the user creates at least one model using their relation in our sandbox environment which is discussed in the following sections. The ontology design tool then maps each user generated model to the theories in each selected core hierarchy. First, it converts the models into FOL sentences, and then uses a FOL reasoner to test each UM_i

against each $CH_{m,n}$ in the hierarchy for consistency. It uses a breadth first specialization algorithm, to find the most specific theory which is consistent with UM_i. Having situated itself in at least one core-hierarchy, the algorithm then maps out meaningful spaces and proposes "experiment" models to the user. Falsifying models allows the algorithm to navigate and eventually determine the strongest set of theories in the repository that are consistent with the user's understanding of their idea.

This reliance on such categorized (and hence mapped) axioms of interest is driven primarily because the characteristics of these theorems are often well understood and accessible. This apparent limitation however provides an opportunity for such an ontology repository to also positively contribute back to the logic (i.e. mathematical logic and/or computer science) communities.

Again, we recall that in general, $I = T_i \cap O_i$. The process described above should elucidate the T_i's to a degree of precision corresponding to the detail provided by the repository. While not fully developed in this thesis, a useful feature for a repository would be to elicit from ontology designers the O_i they intend as well. For example, there may be particular interest in trees where each node has exactly three leaves system. By collecting and collating these O_i for each user theory, a histogram might be developed, based either on frequency, or perhaps a to-be-defined measure of utility for each.

Users interacting with the ontology repository would then have the option to interact with the ontology learning process based on either the accepted mathematical hierarchy, or one that incorporates frequent/useful user generated classes as well. In this way, at the expense of expanding the search base, the axioms generated by the process will be more specific and have greater resolution. Additionally, these new O_i would suggest new classes of structures that theoretical mathematicians may wish to investigate the properties of further.

Another area for further development, as noted above in the Combining Hierarchies and Lemma 4-4, there is a case where every user generated model maps into more than one

core-hierarchy. If these hierarchies are inconsistent, then the algorithm as it stands cannot provide the set of strongest axioms from the repository. In order to extend the algorithm to this case it is necessary to flesh out the interactions between the hierarchies, perhaps at a meta-theoretic level. Currently, this problem is outside of the scope of this thesis.

Finally, this approach need not only be applied at the level of mathematical structures. Recalling Figure 3-2 in chapter 3, domain ontologies for Time, Space, Mereology etc. could also be used to define a relation or function. While those ontologies themselves might be built upon these same logical building blocks, mapping into theories at that level of abstraction may be more natural or fruitful in some cases. In general, this approach would work for whichever level of abstraction one might wish to implement.

4.3 Limitations

A recap of the limitations of this approach is explored here. Firstly, while we do not require ontology designers to be well versed in FOL, axiom generation for relations depends on the existence an accessible and interpretable form for their models. To this end, we must explore first the nature of relations, which is dependent on their arity.

4.3.1 Models

Unary relations correspond to simply naming something and are not of particular interest here. Binary relations correspond to a significant number of relations in use, and are easily expressible as graphs, where the relation is an edge, and its arguments are nodes. The current implementation of this ontology repository, limited to partial orderings makes full use of their representation as graphs or in the case of posets, Hasse diagrams.

Ternary are a bit more difficult, yet still widely accessible in visual form. In general, there are two types of ternary relations:

1. Contextual 2-D relations, where if we have R(a,b,c), a is a contextualizer for b and c (i.e. a is a plane in which b and c interact).
2. True 3-D relations, where R(a,b,c) is best described as three dimensional object

Quarternary relations are more difficult still, however in visual form they may possibly be viewed as a 3-D object undergoing change over some interval. Higher order relations are even less frequent in logical systems, and it is unclear how they might be visualized or manifested in some accessible model form. While the language above relied heavily on the *visualization* of models, it must be noted that in general, this algorithm works for any *manifestation* of a model, whatever sense humans or agents use to primarily relate to it.

Finally, it is impossible to generate diagrams for models of infinite size, say for example, the Real numbers. Similarly, it is also difficult to generate diagrams that represent infinite models, though this side of the problem may be mitigated by adopting a convention where say, "…" represents a continuing infinite sequence that perhaps repeats a pattern being shown. Similarly, one may show "snippets" of an infinite model, and allow a user to traverse it as desired (i.e. a fractal).

4.3.2 Recognizing Models

Contingent on the above is also the fact that ontology designers should be able to discern whether any given model is consistent with user of their relation or not. As the number of nodes included in the model increases, it may become increasingly difficult for agents to be able to accept or reject a particular M^{test}.

4.3.3 Consistency vs Equivalence

Continuing the limitations of an approach relying on models to communicate logical axioms is that the strongest we can test for in general is consistency, not equivalence or isomorphism. In the ideal case, where $I = T_k$, where T_k is specifically one of the recognized mathematical structures, then we may have an equivalence, otherwise we are

left with at most providing a conjunction of T_i, while remaining silent about any possible O_i. Again, we may use this limitation to our advantage and bring theoretical logic and pragmatism closer, but collecting and making note of user generated O_i.

4.3.4 Combining Theories

As noted in the pages above, the algorithm is unable to provide an answer if the set of theories in the repository that are consistent with the user feedback are inconsistent amongst themselves. There are several possibilities here, firstly the user may actually want to say that the relation R, is *either* A *or* B *or* C, but not all of them at once. Alternatively, it may be the case that the theories in the repository only accidentally capture fragments of what the user actually wants. There is the final possibility that the user is trying to define too much at once, and the relation or function behaves differently depending on what its domain is (sorts). Unfortunately, at this point in time, the algorithm cannot distinguish between these cases and simply terminates.

4.3.5 Reasoning Time

The final component of this endeavour which merits closer, empirical inspection is the potential reasoning time to both construct models, and to test models for consistency. Testing models for consistency is restricted to those inputted by the user, and in general as the number of nodes the user uses increases, the greater the reasoning time. However, these models are finite and also limited by the complexity that the human ontology designer is satisfied with. For most cases of models generated, it is conjectured that the reasoning time to map the user to a starting point in a given Core Hierarchy is not a limiting factor.

Model generation is also proportional to the number of nodes desired. This number is left to users, with warnings given if they wish models to be generated with > X nodes. The complexity and precise number of X is governed by the particular core hierarchy under consideration. Once a model is generated by a program such as MACE, in the case of posets, or any binary relation, it is straightforward to convert the resultant matrix to graph

form. However, MACE still has difficulty generating interesting models for certain classes of theories, this may be due to the fact that it is primarily a sat solver. Moreover, as mentioned above, for higher arity relations, even if an interesting model has been generated, generating a visual representation is not always straightforward, and may require more intensive processor power as well.

4.4 Use Cases

Two uses cases are presented below to illustrate the algorithm in action. Unfortunately, since only one core hierarchy has been developed, the case where user models map into more than one core hierarchy does not have an example. The first use case shows how one might have gone about defining the PSL *subactivity* relation as it applies to atomic subactivities (PSL 2007). In this case, the algorithm is able to provide the user with *exactly* the axioms she desired (i.e. $O_i = \perp$).

The second use case develops axioms for a *flows* relation, as taken from "geography.kif" from SUMO (SUMO 2004). There are no actual axioms to compare it to except for the generic poset axioms which are included in the original SUMO formulation. Since there is no real user here, the author acts as the discriminating agent. The axioms derived for *flows* turn out to be those for a meet semilattice, which is a *stronger* theory than those of the original *poset* in SUMO.

4.4.1 PSL – Subactivity (over atomic activites)

The sub-activity relation restricted to atomic activities as defined in PSL corresponds to distributive, join-semi-lattices. The relevant set of axioms, as exposited by NIST is as follows:

Axiom 8 The semilattice of atomic activities is distributive.

(forall (?a ?b0 ?b1)

```
(if    (and   (atomic ?a)
              (atomic ?b0)
              (atomic ?b1)
              (subactivity ?a (conc ?b0 ?b1))
              (not (primitive ?a)))
       (exists (?a0 ?a1)
              (and   (subactivity ?a0 ?a)
                     (subactivity ?a1 ?a)
                     (= ?a (conc ?a0 ?a1))))))
```

Equation 4-2

We will instead generate models for the subactivity relation and follow along with the ontology design tool. Ideally, the resultant axioms from the repository will correspond to a join-semi distributive lattice.

Step 1 – User Generated Models

For the purpose of this illustration, we assume that the user does not know these axioms, only the sense in which the relation should be used, i.e. candidate models. Thus, the first step in the algorithm would be to elicit a model from the user, as in Figure 4-10. The user would use the Sandbox tool to "draw" models for their relation. In the sandbox mode, basic properties for the relation may be denoted, such as Reflexive, Transitive and Anti-Symmetric. Here the user provided three models, UM1, UM2 and UM3.

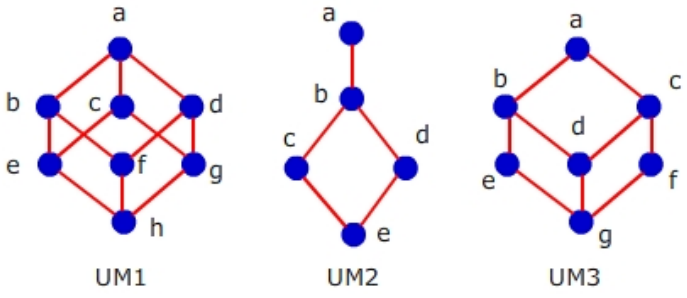


Figure 4-10 – User Models - Subactivity

Step 3 – Generate Complete Diagrams

Each of these is decomposed to a series of statements that can be fed into a reasoner. Every node is labeled and a generic relation, *Rel*, is applied. For example, the complete diagram of UM2 would be the following statements:

(exists (a b c d e)
(and
(R a a) (R a b) (R a c) (R a d) (R a e) (R b b) (R b c) (R b d) (R b e) (R c c) (R c e) (R d d) (R d e) (R e e) –(R e d) –(R e c) –(R e b) –(R e a) –(R d b) –(R d a) –(R d c) –(R c d) –(R c b) –(R c a) –(R b a)))

Equation 4-3

It should be noted that these diagrams may have an ambiguity that is actually resolved by convention as determined by the user in the Sand Box tool. Namely, the user specifies whether each line is "Reflexive" or "Transitive." Symmetry is disambiguated by either checking the corresponding box, or using arrows with points (both options available in the Sand Box tool).

Step 4 – Map to Core Hierarchies

Each of these models are then compared against the hierarchies in the repository using a breadth-first specialization algorithm and an automated reasoner, in this case, Prover9. There are two ways to test for consistency, the axioms of each theory alongside the above statements alongside the mapping statement:

(forall (x y) (iff (lte x y) (R x y)))

Equation 4-4

may be tested for consistency by searching for a contradiction. This scenario involves waiting for the theorem prover to exhaust its search space. Alternatively, the theories in the repository alongside the generated statements may be used to construct a model (which should result in the same model that the statements represent). If such a model is obtained then the theory was consistent with the model.

Step 5 – Check if Algorithm Applies

Using the latter approach we find that UM1 is consistent with all theories in poset up to Boolean Lattice, UM2 maps into distributive lattice and UM3 maps into join-semi-distributive lattice. It should be noted that in the following text, the indexes for sets will occasionally use names for readability, though one may imagine a table behind the scene that correlates every name to an integer allowing increments. For example, the extension of $\mathcal{Consistent\_UM}_{3,\text{poset}}$ = {poset, atomic covering property, meet semi-lattice, join semi-lattice, join semi-distributive lattice}.

Step 6 – Initialize in each Core Hierarchy

To begin the second phase of the algorithm, we first construct:

$$\mathcal{Consistent}_{\text{poset}} = \mathcal{Consistent\_UM}_{1,\text{poset}} \cap \mathcal{Consistent\_UM}_{2,\text{poset}} \cap \mathcal{Consistent\_UM}_{3,\text{poset}}$$

In this case, it is the same as $\mathcal{Consistent\_UM}_{3,\text{poset}}$ since the others are subsumed by that set. Pruning all parents from the set initializes the algorithm at $T_{0,\text{poset}}$ = join semi-distributive lattice corresponds to steps 6 and 7 above.

Step 10-11 – Generalize Test

The algorithm now initializes three sets, $\mathcal{Propose}_j = T_{0,\text{poset}}$ = join semi-distributive lattice and $\mathcal{Accept}_j = \mathcal{UM}$ and $\mathcal{Reject}_j = \perp$. We also need to explore the models above $T_{0,\text{poset}}$ which are collected in the set $\mathcal{Above}_{\text{poset}}$ which in this case is simply ={bounded join semi-

lattice}. Using an automatic model generator, the software would then produce a model for (BJSL ∪ ~(JSDL))

A resultant such model might be as below in Figure 4-11:

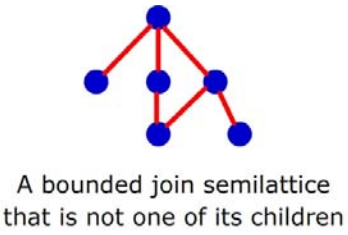


Figure 4-11 – Software Generate Model – Subactivity.

Assuming that the user recognizes that this is an undesired model for the subactivity relation, it would correspond to **Step 11a**. Meaning they would select "Reject" adding this model to $\mathcal{R}eject_{\text{poset}}$ and pruning "join-semi-lattice" from $\mathcal{A}bove_{\text{poset}}$. According to **Step 11c** we then check to see if $\mathcal{A}bove_{\text{poset}}$ is empty, we note that it is and move on to the next step.

Step 11 – Prune Specialize

Here, $\mathcal{B}elow_{\text{poset}}$ consists of only one element, and is the set {Semi-Distributive Lattice}. Consequently, the associated model space is quite straight forward, consisting of only one subspace. The first model to test is JSDL ∪ ~SDL, one such model would be as in Figure 4-12 below:

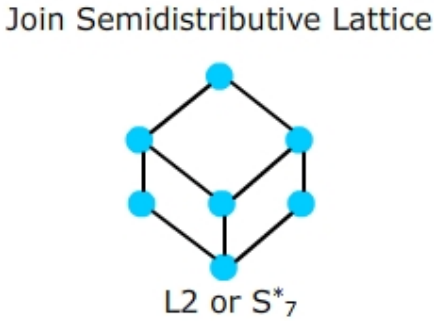


Figure 4-12 – JSDL.

The user recognizes that this is a possible model for their intuition, so they select "Accept" adding this model to $\mathcal{Accept}_{\text{poset}}$ as in **Step 12c**. Since there is only one element in $\mathcal{Below}_{\text{poset}}$, we construct a model for SDL and present that to the user. Presumably they realize that the presented model (corresponding to a *semi-distributive lattices*) is also desirable and so accept that it well (corresponding again to **Step 12c**), thus concluding the "Prune Specialize" steps.

Step 13 – Proposed Axioms from Hierarchy j

At this point, we have completed investigating hierarchy j, which in this case was the poset hierarchy. Since the user did not reject any model in the Prune Specialize steps, no changes were made to $\mathcal{Propose}_{\text{poset}}$, meaning that the axioms presented to the user are those for join semi-distributive lattices, i.e. $T_{0,\text{poset}} = H_{0,\text{poset}} = \text{JSDL}$. The extensions of the accepted and rejected sets are:

$$\mathcal{Accept}_{\text{poset}} = \{\text{UM1, UM2, UM3, JSDL} \cup \sim \text{SDL, SDL}\}$$
$$\mathcal{Reject}_{\text{poset}} = \{\text{BJSL} \cup \sim\text{JSDL}\}$$

where the former are all consistent and the latter are inconsistent with JSDL. We now need to check if there were any other hierarchies and whether we need to combine axioms. In our example there are no other core-hierarchies (since no others have yet been implemented.)

Step 14 – Check for Full Coverage

The final step of the algorithm, would yield the first case (**Step 14a**), namely, where the only $\mathcal{Consistent}_{\text{j}}$ that is non-empty is $\mathcal{Consistent}_{\text{poset}}$. Consequently, the algorithm would state:

> *The ontology design tool has determined that the set of axioms $H_{0,j}$ is collection of the strongest theories from the repository that are consistent*

with all the models you have accepted as being possible manifestations of your relation, and it is inconsistent with all those labeled as Reject.

The user would now be able to use the axioms in the repository for their subactivity relation without having had to formulate the axioms themselves. The only requirements for the user were that they be able to construct at least one model for subactivity as well as recognize whether a proposed model was an acceptable manifestation. Additionally, to generate an unambiguous translation of the diagram into FOL statements for a binary relation, it was necessary to specify whether the relation was transitive, reflexive and anti-symmetric. However this does not require the user to know the axioms for these properties, they simply need to specify how they wish the arrows on the diagram behave.

4.4.2 Flows (Sumo)

The flows relation is defined in Geograph.kif available on the SUMO site (SUMO 2004). Its relevant axioms are defined thusly, and the entire axiom set may be found in the associated file (SUMO 2004):

(instance flows BinaryPredicate)
(instance flows AsymmetricRelation)
(instance flows TransitiveRelation)
(domain flows 1 (ExtensionFn Fluid))
(domain flows 2 (ExtensionFn Fluid))
(domain flows 1 Physical)
(domain flows 2 Physical)
(subrelation tributary flows)
(subrelation flows connected)

(documentation flows EnglishLanguage "(&%flows ?FLUID1 ?FLUID2) means that the
&%Physical ?FLUID1 moves towards the &%Physical ?FLUID2, to which

```
it is &%connected.")

(=>
        (flows ?FLUID1 ?FLUID2)
        (connected ?FLUID1 ?FLUID2))

(=>
        (flows ?FLUID1 ?FLUID2)
        (orientation ?FLUID1 ?FLUID2 Upstream))

; (=>
;       (and
;               (flows ?STREAM ?WATER)
;               (origin ?STREAM ?SOURCE) ;; bogus - domain 1 violation
;                       (orientation ?WATER ?SOURCE ?DIRECTION))
;       (downstream ?STREAM ?DIRECTION))
```

Equation 4-5

In its current form, the flows diagram is simply a strict partial order (no reflexivity). The following exercise will construct three models for the flows relation that adhere to the axioms listed above based on real life rivers. The algorithm described above will then try to generate axioms that might be more expressive and appropriate as vetted by a human (in this case me).

The rivers used for generating models are the Humber River, the Toronto Harbour and the Trent River as shown in Figure 4-13.

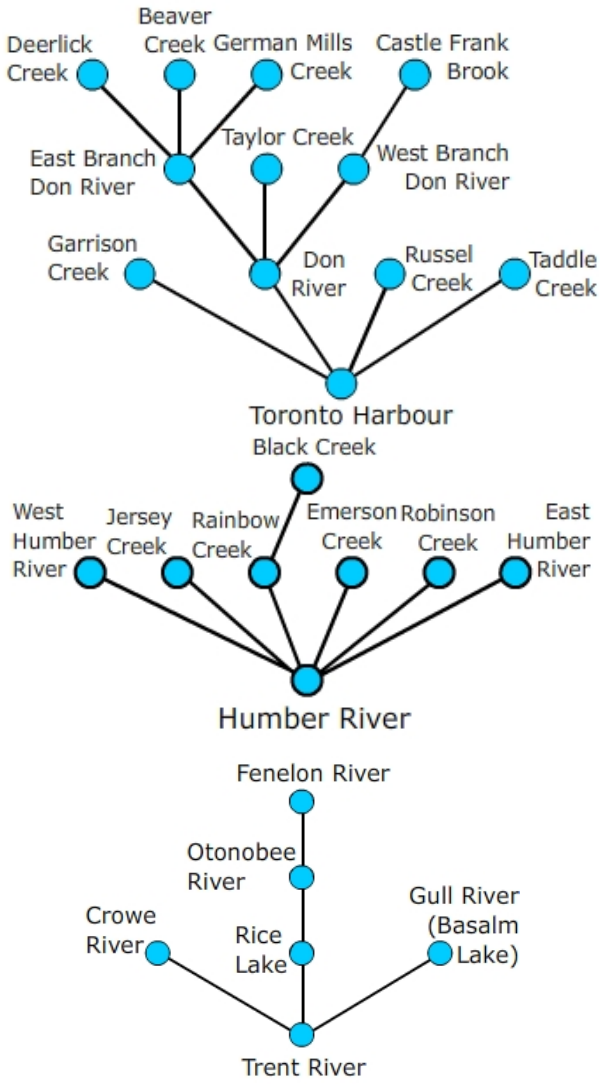


Figure 4-13 – User Models – Flows.

The corresponding complete diagram for the Trent River would be (note: for brevity, the names in the diagram have been abbreviated below, i.e Trent River = TR):

(exists (TR CR RL OR GR FR)

(and

((R FR FR) (R FR OR) (R FR RL) (R FR TR) (not(or (R FR CR) (R FR GR))) (R OR OR) (R OR RL) (R OR TR) (not (or (R OR CR) (R OR GR) (R OR FR))) (R RL RL) (R RL TR) (not (or (R RL OR) (R RL FR) (R RL

CR) (R RL GR))) (R CR CR) (R CR TR) (not (or (R CR RL) (R CR OR) (R CR FR) (R CR GR))) (R GR GR) (R GR TR) (not (or (R GR CR) (R GR RL) (R GR OR) (R GR FR))) (R TR TR) (not (or (R TR CR) (R TR GR) (R TR RL) (R TR OR) (R TR FR)))

))

Equation 4-6

Map to Core Hierarchies, Rid Parents

The strongest theory which is consistent with the user generated models in this case is Down-Tree. Hence, $\mathcal{Consistent\_UM}_{\text{i,poset}}$ contains every theory above Down-Tree. The intersection of each $\mathcal{Consistent\_UM}_{\text{i,poset}}$, followed by the pruning of parents yields $\mathcal{Consistent}_{\text{poset}}$ = {Down-Tree}. Since there is only one element in $\mathcal{Consistent}_{\text{poset}}$, we have **Case 1** corresponding to **Step 8.**

Generalize Test

We now construct the set $\mathcal{Above}_{\text{poset}}$ for Down-Tree which is {Down-Forest, Meet-Semi Lattice}. The first model is constructed for Down-Forest U ~(Down-Tree) as shown in Figure 4-14.

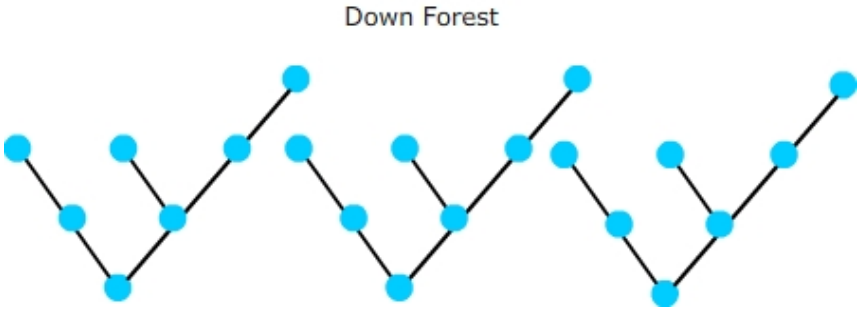


Figure 4-14 – Down Forest

Since I am acting as the user in this case, my understanding is that most uses of flow have every river or body of water flowing into a unique body of water. Hence, the model is rejected.

The model for Bounded Meet-Semi Lattice U ~(Down-Tree), shown in Figure 4-15, however seems acceptable. While none of the user models captured the idea of one river flowing into two rivers, it seems eminently possible. Indeed, often when a river is wide enough and has an island that is large enough in its midst, it often splits into two, even though eventually, both rivers feed into the same body of water.

Figure 4-15 – MSL.

Hence, the new strongest theory is BMSL. Again, we construct the set $\mathcal{A}bove_{\text{poset}}$ = {Poset}. The model for poset, as shown in Figure 4-16 has no unique bottom and is hence rejected. Since there are no other elements in $\mathcal{A}bove_{\text{poset}}$ we now begin to look at those elements below BMSL.

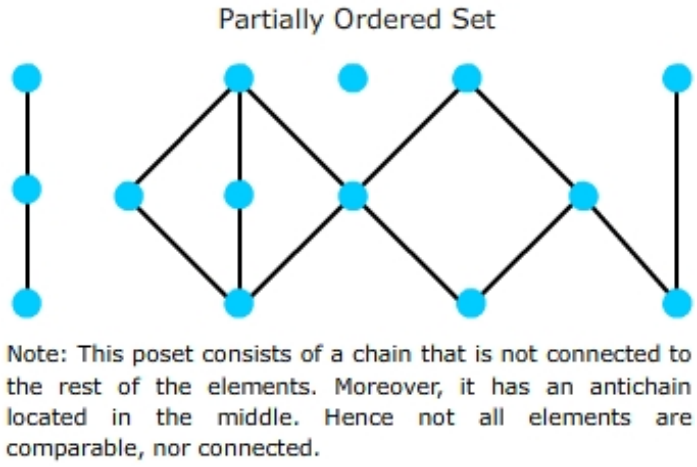


Figure 4-16 – Poset.

Prune Specialize

The set $\mathcal{B}elow_{\text{poset}}$ = {Down-Tree, MSDL, MPCSL, MSML}. We begin by creating a model for a meet semilattice that has none of the properties of its children, i.e. BMSL U ~(Down-Tree) U ~(MSDL) U ~(MSML) U ~(MPCSL) as shown in Figure 4-17. This seems like a plausible model for rivers, so I click accept.

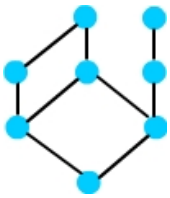


Figure 4-17 – Uniquely a BMSL

Now the algorithm will try each of the extensions of BMSL and see if it can falsify one of them. The first model constructed is DT ∪ ~(MSDL) ∪ ~(MSML) ∪ ~(MPCSL), as in Figure 4-18. Again this seems acceptable, so I select accept. Going through each possible combination of $\mathcal{B}elow_{poset}$, every model seems like a potential model of rivers flowing into one another.

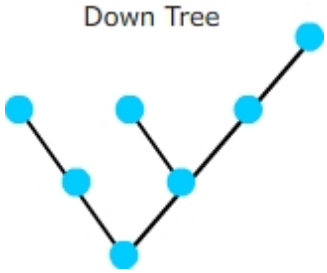


Figure 4-18 – Down Tree

Check Full Coverage and Terminate

As there are no other combinations to try, the algorithm sets $\mathcal{P}ropose_j$ = {BMSL}. We note that after this process"

$\mathcal{A}ccept_{poset}$ = $\mathcal{UM}$ ∪ {MSL ∪ ~BMSL, BMSL ∪ ~DT ∪ ~MSDL ∪ ~MPCSL ∪ ~MSML, MSDL ∪ ~DT ∪ ~MSML ∪ ~MPCSL, MSML ∪ ~DT ∪ ~MSDL ∪ ~PCSL, MPCSL ∪ ~DT ∪ ~MSDL ∪ ~MSML, DT ∪ ~MSDL ∪ ~MSML ∪ ~MPCSL}
$\mathcal{R}eject_{poset}$ = {Down-Forest, Poset ∪ ~MSL}.

The axioms for MSL are consistent with every element in $\mathcal{A}ccept_{poset}$ and inconsistent with every element in $\mathcal{R}eject_{poset}$.

Moreover, since there were no matches in any other hierarchy, the axioms presented to the user for the relation flows would be those for a meet-semi lattice. The output may be presented in two forms, if I as the user had provided the name "flows" for the relation I was trying to define, the axioms for MSL would be presented already in terms of "flows" for "less than or equal to". Otherwise, it would have the original axioms for MSL with the mapping:

(cl-comment "the strongest theory in the repository that is consistent with the relation you are trying to define is Meet Semi Lattice. Below is a mapping from a generic binary relation to those for MSL is provided."

(cl-module Relation)
(cl-import (meet-semilattice))
    (forall (x y)
        (if (Relation x y) (lte x y)
    ))
)

Equation 4-7

The user could then simply use this snippet of CLIF in their ontology, or they could directly paste the meet semilattice axioms locally, simply exchanging the *flows* relation name for *lte*.

Chapter 5

Semantic Mapping Module

5.1 Motivation

One encounters an array of interesting problems when trying to generate and articulate semantic mappings among ontologies, especially in an automated fashion (Pinto et al 1999; Wache et al 2001; Noy 2004). Since ontologies are often developed to address a particular domain, we frequently arrive at the situation captured in the parable about an elephant in a dark room.[15] We encounter the problem where different groups of people "feel" and consequently represent different aspects of the elephant. There may be overlap in what they have sensed and perceived, but the terminology and agreement about they each mean will be quite different. A fundamental question is how to reconstruct the fact that everyone is feeling an elephant? Or more specifically, how may we reconcile the use of diverse names and foci on different aspects of the same concept?

[15] The story consists of a dark room, an elephant and three people who each feel a part of the elephant, and each claim to know what the elephant is. One touching only the trunk describes the elephant as a long hose. Another having felt its legs describes it as a sturdy tree trunk. The last, touching only the tusk, describes the elephant as a smooth dagger.

Semantic mappings attempt to establish how the target formalisms may interact with one another. When there is an overlap in the senses of the terms ontologies are using, we want to know in what ways the semantics are the same, and how terms may be reused. Creating these mappings is an especially acute problem for ontologies developed using description logics since much of the semantics is accessible only outside the system of representation. As Uschold and Gruninger state, in order to fruitfully use an ontology we would like there to be no extra ontological distinctions:

> *Key distinctions are made within the language so that all conclusions can be drawn from the ontology alone. Without this property we would need to represent various assumptions procedurally outside of the language of the ontology. Howeve since one of the purposes of the ontology is to provide a framework for shared understanding if these assumptions are not represented within our ontology we risk a disagreement in how different agents interpret these extra ontological assumptions.* (Uschold & Gruninger 1996, p 25)

This really *is* the problem of semantic mapping, especially acute in description logic ontologies; the same strings or mutually accessible snippets of thought are ambiguous, dependent on the observer. That is to say, when referring to the same represented statements, one may construct multiple acceptable interpretations without knowing how those interpretations are necessarily related to one another – particularly as statements that might be expressible in the language of representation.

Noy provides a good account of the problems and techniques used to address some of the issues in semantic mappings, though the work is largely focused on DL ontologies (Noy 2004). While some strides have been made using techniques such as lexical or structural similarities, looking at instances of terms, property definitions etc, they are often hindered since many DL's also implement different semantics.

It is important to note a fundamental difference here between DL level mappings and mappings based on axiomatized (FOL) ontologies. The latter representations have a richer description of a concept than simply naming it and perhaps stating "Has_A_XXX" relation. While both types of mappings connect relations and/or functions in one ontology to another, FOL expressed mappings will highlight the degree of consistency between the uses of the concepts. This idea of consistency gains greater currency when we have a mirror or prism (i.e. the ontology repository) through which to reflect each target ontology. Since the repository developed here aims to identify the logical substructures that underlie most FOL expressible concepts, mapping then simply becomes recognizing when particular relations or functions reuse the same logical building blocks.

The algorithm developed in this chapter seeks such mappings for FO expressed ontologies. Consequently, it is restricted to providing mappings based on the ontologies in the repository and is silent about any other possible existing links between the target ontologies. However, as noted by many authors, when a ULO exists, the burden of identifying mappings is potentially facilitated, and in this respect our ontology repository acts as an upside-down-ULO. To use the language of Choi introduced in Chapter 2, the semantic mappings being conducted here are those of alignment (Choi et al 2006). Moreover, of the three types of mappings identified, the process described below corresponds to a hybridization of mapping between local ontologies and a globally integrated ontology alongside that of ontology alignment. One difference is that the global ontology used in our case is not necessarily explicitly integrated into the target ontologies. Instead it provides a reference, where images of the ontologies to be mapped are captured and then compared to one another.

5.2 Scope of the Mappings

We must now expand on the nature of what the repository captures, and what the mappings between ontologies mean. First, we note that many of the core-hierarchies and theories in the repository are elaborations of particular relations, functions or combinations there of. For example, the poset core hierarchy is the extension of a type of

binary ordering relation. These specifications of a particular relation or function then allow us to define in what sense ontologies are being mapped to one another through the prism of the repository. Thus, the poset hierarchy allows semantic mappings (if they exist) to be developed between any binary or binary projectable relations in the target ontologies.

The statement above introduces an important point which shows the power of the algorithm, though it requires further elaboration. There are many instances where n-ary relations are mappable to a k-ary relation where n > k. For example, the (min_precedes s1, s2, a) relation in PSL is a ternary relation, which may also be expressed as a binary ordering relation over *s1* and *s2* in the context of *a*. Producing such a projection of a relation allows one to compare it directly to the binary relations in the repository and involves the following straightforward step:

(if (min_precedes s1 s2 a)) (exists (a) (min_precedes_new s1 s2 a)))

Equation 5-1

Holding *a* "constant" (analogous to taking partial derivatives)[16], "min_precedes_new" may now be compared to all the binary relations in the repository. In general, whenever we want to create such a projection, we may do so using the following template:

Let RT be the target relation to be projected. Further, let m = the arity of RT and x1, x2, …, xm its arguments. Additionally, let n = arity of the relation to be compared to in the repository, RO. Then k = m – n and we can create a projection thusly:

(iff (RO y1 … yn) (exists (*k arguments*) (RTnew x1 … xm)))

Equation 5-2

[16] They are analogous in the following sense: for a function, f(x,y,z), if we take the partial derivative of y with respect to x, we are doing so in the context of a particular z (holding z constant). Similarly, when constructing a binary projection for a ternary relation, we hold one of the arguments "constant."

While we can thusly project any n-ary relation to a lower arity relation in the repository, the question also arises which relations do we want to do this for? More generally, how does one decide which relations in target ontologies to compare to one another?

The conventional answer to the above has been to require a human to explicitly state which relations to connect to one another, or perhaps to look at the instance level data to drive the connection. While having a human pick which relations to compare, and/or which terms in an n-ary relation to hold constant and which to map into the repository would facilitate the semantic mapping process, it is strictly, unnecessary (though for efficiency and tractability purposes it is highly desirable).

When presented with a number of target ontologies, assuming no further human intervention (i.e. someone to identify which relations to match, which terms to project over etc, which core-hierarchies to search), the pre-selection algorithm will do two things. First, depending on which core-hierarchies it is searching (since some elaborate binary, ternary relations or function etc.), it will identify all relations in the target ontologies that have a valence equal to or higher than the relations of the particular hierarchy. For every higher order relation in the target ontologies, it will then create all possible projections, i.e.:

(R x y z) becomes:

(exists (x) (Rnew x y z)) OR (exists (y) (Rnew x y z)) OR (exists (z) (Rnew x y z))

Equation 5-3

Next, for each such identified relation, it will posit as an equivalence:

(iff (R_Repository y1 … yn) (exists (k terms) (Rnew x1 … xm))))

Equation 5-4

It will then construct an image for every mapping for each target ontology. The overlay of all the combination of images will then show which relation(s) in which ontologies reuse the same logical patterns, thus yielding the desired semantic mapping.

Once these potential mappings to test have been identified, the main component of the algorithm kicks in. Figure 5-1 to the right illustrates the schematic for the semantic mapping process at a high level. First, a number of target ontologies and corresponding relations to test are fed into the system (i.e. T1-T3). Next, a breadth-first specialization algorithm constructs an image of each target ontology in the repository using a FOL reasoner (i.e. Prover9). Finally, the images of each of the target ontologies are then compared according to combinations specified by the user, yielding where they share the same axioms for concepts and where they differ. It should be noted that this process produces mapping not just between domain ontologies, but makes explicit *metaphorical* mappings, where two ontologies might be using the same logical building blocks to represent distinct phenomena. The next section discusses the elements of the algorithm in greater detail, alongside a correctness proof.

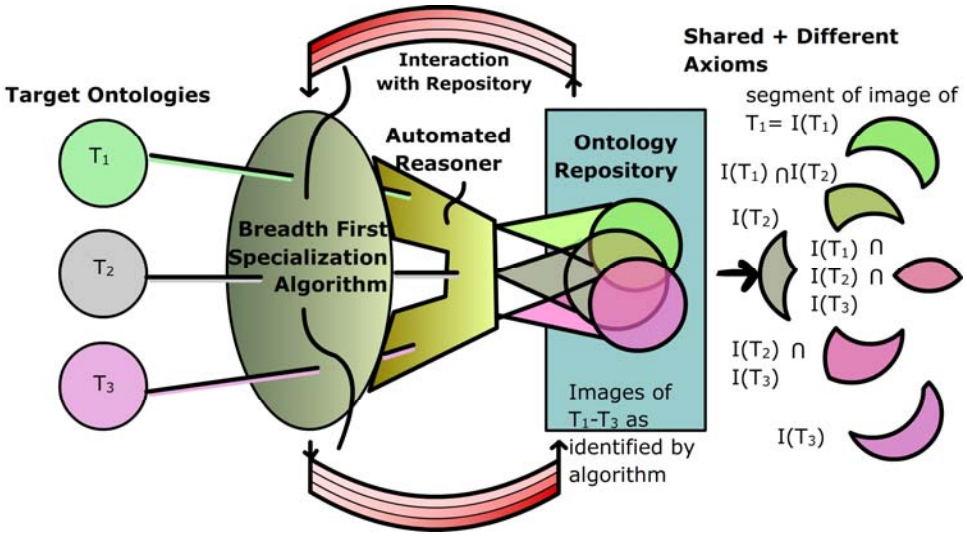


Figure 5-1 – High Level view of Semantic Mapping

5.3 The Algorithm

5.3.1 Elements of Algorithm

An issue that dissuades some from using first order logic to represent their understanding of the world is that there often exist no clear guidelines or templates for how an axiom should be expressed. As noted earlier in this work, the expressivity of FOL allows one to represent the same concept as either a relation or a function, as was the case with meet and join compared to maximal-lower-bound and minimal-upper-bound. Part of the problem in semantic mapping is in knowing that these two different representations refer to the same concepts.

The algorithm developed here is unable to wholly automate that process (nor is that the intent). Instead, whenever multiple articulations of the same logical constructs are noted in literature, every effort is made to store both accounts in the repository and have an explicit manual mapping between them. This is an important point, since it highlights the differences between the approach developed in this thesis and the predominant paradigm in the ontology community at the moment.

Recalling the importance of metaphors in how many of our thoughts and understandings of the world are articulated, the nature of mapping becomes slightly altered (Lakoff & Johnson 1980; Danesi 2002; Pinker 2007). Whereas traditional ontology mappings attempt to generate mappings based on various lexical, (instance) structural etc. approaches, the mapping developed here approaches the problem from the other end. What is being built in the repository is the analog of the "math.h" library for the C family of languages.

Moreover, since the ontologies being considered here are formal and axiomatized, it is logical structures that are of interest, not necessarily what the particular data (extensions of relations, functions etc.) or lexical meanings of the names (strings) are. This means that the mappings already *exist* as represented by the repository. What is left to be done is to identify and specify which mappings apply and where. For example, a subset of

axioms for "On" and "Under" might be the same, implying an important connection between the two concepts. Similarly, permeability in cell membranes and resistance for a resistor may share many similar axioms, which again would be explicated by this process.

Once the logical theories of interest have been captured by the repository, the automated side of the mapping can now search any number of target ontologies, who may have expressed their understanding either in terms of functions or relations and still be able to provide a mapping.

The process for this algorithm is illustrated in Figure 5-2 below. Once the relations to be mapped have been identified, it consists of two parts: first, capturing an image of the target ontology(ies) as it is reflected in the ontology repository; then taking the intersection of the images and in such a way identifying where the targets reused the same logical structures. The mappings already exist in the repository, since they correspond to classes of axioms. Thus, by comparing the images of all the target ontologies, one may highlight how they relate to one another (pair-wise, all-together, etc. – it is at the discretion of whomever is conducting the mapping). This comparison generates both similarities and difference between the targets.

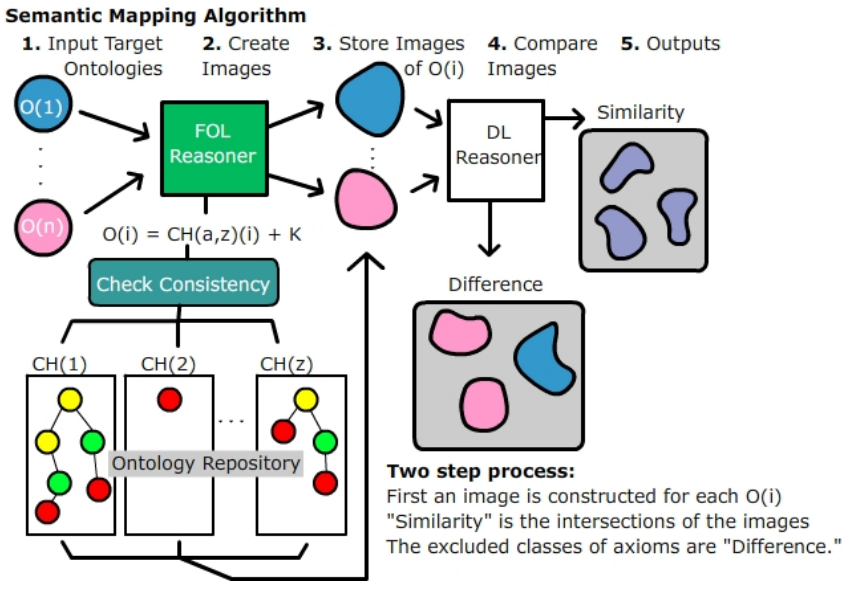


Figure 5-2 – Semantic Mapping Algorithm Flow

In greater detail, the first part requires that an ontology, O_i send the repository its axioms as well as the relation or function to be investigated. For each selected core hierarchy, the algorithm scours the repository using a breadth-first specialization algorithm to test each class of axioms with O_i for consistency. If O_i is inconsistent with a given class, that class and all of its descendents are pruned from the search space. Classes which are consistent are collected in a list, O_i^{Image}. For every core hierarchy with at least one consistent class, each of the strongest classes that were consistent with O_i are noted and collected in the set $O_i^{Image\_St} = \{CH_{m,n}\}$. We recall that as defined in Chapter 4, strongest here refers to theories from which the most can be proven; if A is a non-conservative extension of B, then A is stronger than B. The same image construction process occurs for the remaining supplied target ontologies.

The second component then highlights the semantic mappings. If ONT is the set of the target ontologies, then the intersection of all O_i^{Image} yields the "similarity" between them. "Differences," are a bit subtler, and requires an ontology of reference. We can nonetheless claim that $O_1, O_2, \ldots, O_i$ are consistent (if at all), up to $Sim(ONT) = \{CH_{m,n}\}$, where $Sim(ONT) = O_1^{Image} \cap O_2^{Image} \cap \ldots \cap O_n^{Image}$ (or any combination thereof). This process is outlined in detail below. In contrast, difference is only useful in a pair-wise fashion.

Let us briefly consider what exactly is being achieved by the process described above. First, we consider the composition of the target ontologies, how they might be reflected through an ontology repository and how mappings may be derived. As noted above, any target ontology will initially be compared to sets of axioms in the repository. We will take the approach that any target ontology O_i is equivalent to:

$$O_i = T_i \cup K$$

Equation 5-5

The first term, T_i captures the set of all theories present in the ontology repository, while K represents sets of axioms that do not correspond to any theory in the repository. When we thus construct a mapping, we can only say something about O_i based on the repository. Thus, by definition, the mapping is silent about any theories not found in the repository.

Our problem now is the relationship between two sets of ontologies. The possible relationships between them are captured in the hierarchy below:

A and B are **equivalent** to one another
A **entails** B *or* B **entails** A
A is **consistent** with B
A and B are **independent**
A and B are **inconsistent**

The strongest case would be to establish equivalence; however, because of the existence of K, we are never guaranteed to find equivalence between target ontologies and those in the repository, let alone between them. The strongest mapping that the process described above can achieve is to establish the consistency of O_i with some union of $CH_{m,n}$. Once this has been done, we can compare the images and construct mapping equivalences between the images and by extension the target ontologies. Figure 5-3 below shows the detailed steps for the algorithm.

Figure 5-3 – Semantic Mapping Algorithm

Semantic Mapping Algorithm in Steps

Inputs:

Axioms for $O_1,\ldots,O_n$ – the target ontologies.
Relation(s) or function(s) to be mapped.

Output:

1. The set of the strongest theories that are consistent with $O_1,\ldots,O_n$ and are an extension of a set of theories from the ontology repository.
2. As defined by the user, sets of theories in the repository that are consistent with at least one of the O_i but not consistent with at least one of the other target ontologies.

Steps:

Algorithm 1 – Create an Image

1. **Get Axioms:** For every O_i, send axioms and relation to map to repository
2. **Initialize Image:** Let O_i^{Image} and $O_i^{Image\_St}$ be sets in which we will store the consistent theories in the repository for O_i
3. Let $CH_{m,n}$ be the m^{th} theory in core-hierarchy n
4. For each n, ranging over all m's, starting at each root begin breadth-first specialization matching
5. **Check Consistency:**
    a. If $CH_{m,n}$ is consistent with O_i
        i. then store it in O_i^{Image} and $O_i^{Image\_St}$
        ii. if $CH_{m,n}$ is a descendent of element $CH_{m-k,n}$ also in $O_i^{Image\_St}$,
            1. then remove $CH_{m-k,n,}$ from $O_i^{Image\_St}$
    b. else remove $CH_{m,n}$ and its children from the search space
6. Let *NoMap* be the set defined thusly if for all hierarchies, there is no $CH_{m,n}$ that is consistent with O_i, add O_i to *NoMap*.
7. If *NoMap* is not empty
    a. Display the following to the user: "Target Ontologies O_a, O_b, O_c etc. do not map into the repository. If these ontologies are included in the construction of *Sim*(ONT*) and *Diff*(O_i, ONT*) no mappings will be found."

Algorithm 2 – Articulate Mappings

8. **Define Sim:** Let user define $\mathcal{ONT}^*$, as the set of ontologies (O_1, …, O_m) for which to construct a mapping
9. Let $\mathcal{Sim\_All}(ONT^*) = \mathcal{Sim}(ONT^*) = O_1^{Image} \cap \ldots \cap O_m^{Image}$
10. **Strongest Sim:** For every element A ϵ $\mathcal{Sim\_All}(ONT^*)$,
    a. If there exists an element B ϵ $\mathcal{Sim}(ONT^*)$, where A is an extension of B according to the repository
        i. then remove B from $\mathcal{Sim}(ONT^*)$
11. **Define Diff:** Let user pick the O_i and $\mathcal{ONT}^*$ for which to construct a difference
12. Let the $\mathcal{Diff}(O_i, ONT^*)$ be $O_i^{Image\_St}$ / $\mathcal{Sim}(ONT^*)$
13. Repeat **Define Sim – Define Diff** as desired for user.
14. Present the user with each $\mathcal{Sim}(ONT^*)$ and $\mathcal{Diff}(O_i, ONT^*)$

5.3.2 Correctness Proof

Proving the correctness of the algorithm requires first proving three lemmata corresponding to creating an image for each target and then constructing mappings based on intersections of the images. In the ensuing sections, the following notation will be used:

$CH_{m,n}$ refers to theory m located in core-hierarchy n.
$O_i = O_1, \ldots, O_k$ refers to one of the target ontologies
$\mathcal{ONT} = \{O_1, \ldots, O_k\}$ is the set of all the target ontologies

Lemma 5-1 will show that the breadth-first specialization algorithm will yield the desired image for each O_i. Lemma 5-2 establishes that a set with the strongest theories from the repository may be constructed. Lemma 5-3 shows that sets corresponding to a sense of similarity and difference may be constructed. Finally, Theorem 5-1 proves the correctness for the algorithm.

Lemma 5-1 (Construct Image)

For any target ontology O_i, one may construct a set, O_i^{Image}, which exhibits the following properties:

1. *the members of O_i^{Image} are exclusively $CH_{m,n}$ or $\perp$.*
2. *every $CH_{m,n}$ in O_i^{Image} is consistent with O_i.*

Proof. We begin by noting that property 1 is trivially satisfied because we are constructing the image only out of elements arising from the repository.

The algorithm stores $CH_{m,n}$ in O_i^{Image} if and only if $CH_{m,n}$ is consistent with O_i according to a reasoner. This ensures that a class of axioms is placed in O_i^{Image} only if it is consistent with the target ontology, thus satisfying property 2.

Now, since in any core hierarchy, each theory is an extension of another, we can start with the most general class of axiom, exhaust classes at that level and then consider the children of these theories. Specifically, if $CH_{m,n}$ is consistent with O_i we add that theory to the set of theories consistent, O_i^{Image}. If it is inconsistent, we prune it and all of its descendents.[17]

Thus every theory in the set O_i^{Image} is consistent with O_i and from the repository. **QED.**

Lemma 5-2 (Strongest Image)

Let O_i^{Image} be the set of theories composed of $CH_{m,n}$ that are consistent with O_i. We can then construct the set $O_i^{Image\_St}$ contained in O_i^{Image} that is the set of the strongest theories $CH_{m,n}$ that are consistent with O_i.

Proof. The design of the repository is such that every core-hierarchy is a join-semi-lattice. There is always one root for all $CH_{m,n}$ in hierarchy n. As we are constructing

[17] We may prune its descendents because we are using a monotonic logic, were it non-monotonic, the algorithm would require some modifications.

O_i^{Image} we simultaneously construct $O_i^{Image\_St}$. Whenever a theory, $CH_{p,n}$ is added to O_i^{Image} we also add it to $O_i^{Image\_St}$, additionally, if $CH_{p,n}$ is an extension of $CH_{q,n}$, then we remove it from $O_i^{Image\_St}$. Since the "genealogy" of every class is explicitly stated with respect to a core-hierarchy, the only elements in $O_i^{Image\_St}$ are those $CH_{m,n}$ that have no children, corresponding to our definition of strongest above, since they are those theories that "say the most about the world."

Consequently, we have ensured that only the strongest, most specialized theories $CH_{m,n}$ from the ontology repository are contained in $O_i^{Image\_St}$ and are consistent with O_i. **QED.**

The final element we need for the correctness proof is Lemma 5-3(Image Comparison). It shows that for any set of theories, we can construct resultant sets, $\mathcal{Sim}$(ONT*) and $\mathcal{Diff}(O_i$, ONT*) which correspond to a particular sense of "Similarity" and "Difference."

Lemma 5-3 (Image Comparison)

Given twp sets of theories, O_i^{Image} and $O_i^{Image\_St}$ there exists sets of theories, $\mathcal{Sim}$(ONT) and $\mathcal{Diff}(O_i$, ONT*) that satisfy the following properties:*

1. *Every element in $\mathcal{Sim}$(ONT*) is also an element in every O_i^{Image} where $O_i \in ONT^*$.*
2. *Every element in $\mathcal{Sim}$(ONT*) is the strongest theory in the repository that is consistent with every $O_i \in ONT^*$.*
3. *Every element $CH_{m,n}$ in $\mathcal{Diff}(O_i$,ONT*) is also an element in $O_i^{Image\_St}$ and not found in at least one O_j^{Image}, where $O_j \in ONT^* \setminus O_i$.*

Proof. From Lemma 5-1 we know that every element in both $\mathcal{Sim}$(ONT*) and $\mathcal{Diff}(O_i$,ONT*) is a theory from the repository since they are constructed using sets adhering to Lemmata 5-1 & 5-2.

If we take $\mathcal{Sim}$(ONT*) to be the intersection of all O_i^{Image}, where $O_i \in ONT^*$, we have satisfied property 1. Step **Strongest Sim** in the algorithm above ensures that only the

strongest theories are stored in *Sim*(ONT*). This is achieved because the algorithm checks every element in *Sim_All*(ONT*) and if A and B are both in *Sim*(ONT*) and A is an extension of B, then it removes B. Were there a stronger theory not in *Sim_All*(ONT*), we would have a contradiction because there would be a stronger theory that was consistent with O_i, but not in O_i^{Image}.

Next, let *Diff*(O_i, ONT*) = $O_i^{Image\_St}$ \ *Sim*(ONT*), thereby removing all the elements in $O_i^{Image\_St}$ that were consistent with every other O_j^{Image}. We have thus satisfied property 3 since the only elements left in Diff(ONT*) are those that are consistent with O_i and inconsistent with at least one O_j, where both O_i and O_j ϵ *ONT**. **QED.**

Theorem 5-1 (Semantic Mapping Algorithm)

Let Sim(ONT) be the semantic mapping between the ontologies in ONT* and Diff(O_i, ONT*) be the differences between the target ontologies to be mapped. The outputs will then exhibit the following property:*

1. *Every theory $CH_{m,n}$ in Sim(ONT*) is the strongest in the repository that is consistent with every $O_i \epsilon$ ONT**
2. *Every theory $CH_{m,n}$ in Diff(O_i, ONT*) is consistent with $O_i \epsilon$ ONT* and $CH_{m,n}$ is inconsistent with at least one $O_j \epsilon$ ONT*, where $i \neq j$.*

Proof. By **Lemma 5-3**, we know that every theory in *Sim*(ONT*) is also a member of O_i^{Image}, where O_i ϵ *ONT**. Additionally, the step **Prune Sim** in the algorithm above ensures only the strongest theories are stored in *Sim*(ONT*). Further, by **Lemma 5-1** we also know that each element in O_i^{Image} is a theory from the repository that is consistent with O_i. Together, we have shown that *Sim*(ONT*) consists of the strongest set of theories from the repository that are consistent with each O_i ϵ *ONT** as required, thus satisfying property 1.

Similarly, by **Lemma 5-3**, we know that every theory in $\mathcal{Diff}(O_i, ONT^*)$ is a member of $O_i^{Image\_St}$ and not a member of at least one O_i^{Image}, where O_i and $O_j \in ONT^*$. By **Lemma 5-1** again, we know that every $CH_{m,n}$ in O_i^{Image} is consistent with O_j. Finally, by **Lemma 5-2,** we know that if $CH_{m,n} \in O_i^{Image\_St}$ then it is the strongest theory in the repository that is consistent with O_i. Thus we have satisfied property 2. **QED**.

5.4 Use Case

To illustrate this algorithm, let us attempt to generate a semantic mapping for the following three relations:

1. (subactivity a1 a2) from PSL
2. (flows f1 f2) from SUMO

Let O_1 = PSL-Core + Subactivity + Atomic Activities (PSL 2007), O_2 be the axioms for *flows* as defined in (SUMO 2004) and refined in the previous chapter. We now have defined our two target ontologies and the relations to explore; the next step is to construct an image for each target.

5.4.1 Image for Subactivity

The full axioms under consideration for the *subactivity* relation are available on the PSL website (PSL 2007). First, the axioms must be translated into a language readable by whichever ATP is implemented, in this case, it is Prover9. Next we need to identify which relation to map, here we will use (subactivity x y) and will be comparing it to (lte x y) having restricted ourselves to the only core-hierarchy currently available

Thus the mapping statement is:

```
(forall (x y)
        (iff     (lte x y)
```

(subactivity x y)))

Equation 5-6

As illustrated in Figure 3-5 several pages ago, the root is simply the axioms for poset. Thus the breadth-first algorithm will compare O_1 with $CH_{1,poset}$ by inputting both sets of axioms including the mapping relation into Prover9 as shown in Figure 5-4. Consistency

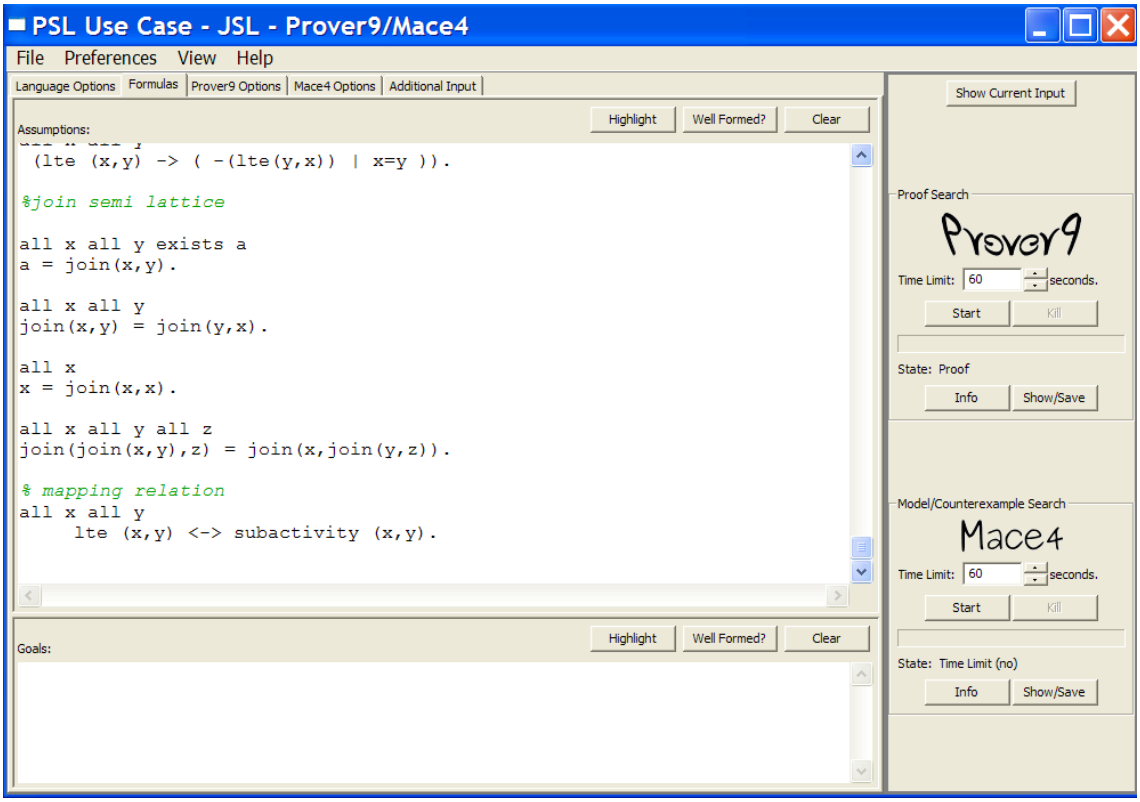


Figure 5-4 – Prover 9 Screen Shot

may be ascertained in two ways: one may either run a "Proof Search" or use Mace4 to construct a model. For this use case, Mace4 was used to generate models to test consistency.

Now that consistency with Poset has been established, we add $CH_{1,poset}$ to O_1^{Image} and $O_1^{Image\_St}$ and then navigate down the poset hierarchy. The next theory is that of the "atomic-covering property." Once again, it is consistent, thus we add it to O_1^{Image} and $O_1^{Image\_St}$ while removing $CH_{1,poset}$ from $O_1^{Image\_St}$.

The next level of the hierarchy has three theories, that of Ortholattice, Meet-Semilattice and Join-Semilattice. Of these, the first two are inconsistent so they and their children are pruned from the search space. Join-Semilattice is however consistent, and thus it is added to both O_1^{Image} and $O_1^{Image\_St}$, while$CH_{2,poset}$ is removed from $O_1^{Image\_St}$.

The level below would have two theories to investigate, however Lattice was pruned since Meet-Semilattice was inconsistent. Testing the (lte x y) relation as extended by the Join-Semidistributive Lattice theory yields a consistent result with (subactivity a1 a2). As before, we add that theory to both images, taking care to remove its parent from $O_1^{Image\_St}$. The next theory to investigate according to the hierarchy is that of Distributive Lattices. Testing the axioms of this theory against O_1 shows that it is inconsistent, thus we prune it and its children.

At this point there are no other elements in the search space for the poset hierarchy, concluding the breadth-first specialization approach for O_1. The two sets thusly constructed are O_1^{Image} = {poset, atomic covering property, join-semi-lattice, join-semi-distributive-lattice} and $O_1^{Image\_St}$ = {join-semi-distributive-lattice} corresponding to all and the strongest theories in poset that were consistent with O_1 respectively. Now we move onto constructing an image for the other two relations.

5.4.2 Image for Flows

For the purposes of this use case, we will use axioms for *flows* as developed by the ontology design tool in the previous chapter. Since *flows* then corresponds to the axioms for a meet semilattice that is what the image of *flows* would be, i.e. $O_{flows}^{Image\_St}$ = {BMSL}

5.4.3 Similarity and Differences

We can now construct the similarities and differences for the target ontologies. Since we have three ontologies submitted for analysis, there are three possible similarity measures: the similarity between each of with the repository (their image) and the similarity of both together.

The similarities are captured thusly:

$ONT^*_1 = \{O_{subactivity}\}$ $\quad Sim(ONT^*_1) = O_{subactivity}^{Image\_St} = \{JSDL\}$

$ONT^*_2 = \{O_{flows}\}$ $\quad Sim(ONT^*_2) = O_{flows}^{Image\_St} = \{BMSL\}$

$ONT^*_3 = \{O_{subactivity}, O_{flows}\}$ $\quad Sim(ONT^*_3) = \{poset\}$

Differences can only be determined in pairs, where one element is the ontology for which to determine the difference and another is a set of ontologies to which it is being compared to. Since there are only two target ontologies, there are two possible difference measures as listed below:

$ONT^*_1 = \{O_{subactivity}, O_{flows}\}$

$Diff(O_{subactivity}, ONT^*_1) = \{JSL, BJSL, JSDL\}$

$Diff(O_{flows}, ONT^*_1) = \{MSL, BMSL\}$

This means that the subactivity relation has theories for of join-semilattice, bounded join semilattice and join semi-distributive lattice that are inconsistent with at least one ontology in ONT^*_1. Conversely, flows utilizes meet-semilattice and bounded meet-semilattice, whereas at least one ontology in ONT^*_1 does not. We can use this information if we wished to reuse *subactivity* or *flows* in an application.

5.5 Conclusion

In the ideal case where each O_i consists only of unions or conjunctions of theories from the repository, we would have a complete semantic mapping with equivalence (as was the case for *flows*, as defined in chapter 4). In general however, we are limited to what is stored in the repository, and thus silent about equivalence due to the existence of *K* as defined above.

What the mapping module achieves is the explicit specification of the shared logical building blocks between any number of ontologies, as reflected through the repository.

To use an analogy, whereas O_1 and O_2 might be using different wheels, their hubs are the same. Additionally, O_1 and O_2 might be talking about electric circuits and cell permeability respectively, however the logical constructs used in each might be quite similar. The semantic mapping module would make such connections clear. Finally, the Difference lists also illustrate where a particular O_i may have diverged from other formalizations by extending or interpreting a concept in a different way.

It should be noted that as the repository grows, more of the *K* theories, those that are classified as "other" for the moment shall fall under the purview of $CH_{i,j}$, thereby leading to the ideal limit of full mappings (i.e. where *K* tends to $\perp$).

With the architecture of the repository alongside the algorithms for the two modules detailed in this and the previous two chapters, we now turn our attention to the design conceptualization of each of these components. Extensions of each of these approaches will be discussed in the final chapter regarding Future Work.

Chapter 6

Design Conceptualization

Thesis as a Software Artifact

6.1 Interacting with the Repository

Now that the guidelines for the design and maintenance of the repository and the basic structure of the two algorithms have been discussed, the deliverables of this thesis shall be discussed as a software artifact. How will users interact with the algorithms? How will the constituent elements be presented, stored and access? To this end, it is proposed that the ontology repository and its modules be accessible as a web service. Issues regarding scalability and network efficiency will not be discussed here; instead the focus will be on the interactive manifestation of the processes.

It should be noted that this chapter does not try to rigidly define the final form of the repository that is a job more suited to someone versed in developing web environments. Rather, this chapter will specify what is needed for each page, and provide a concrete example to illustrate what is desired. Final design decisions and general web page development lie outside the scope of this thesis. To ensure maximal accessibility, it is

natural to situate the repository online as a website. The site consists of four overarching elements:

- Browsable Repository
- Ontology Design Module
- Semantic Mapping Module
- Additional Features

As the name suggest, Browsable Repository will develop how users will view the ontologies contained in the repository and how they are related to one another. The latter two modules will detail the process by which each algorithm will be utilized. The Additional Features refers to components of the site that explain the philosophy behind it, provide the user with background, contact and guideline information. Additionally, it encompasses the account system for logging in and either making changes to the repository or saving sessions when engaged in one of the algorithms.

The first view that a user will see when accessing this service will be the homepage as illustrated in Figure 6-1. As in Area A, it should include a brief introduction explaining what the site hopes to achieve and what it consists of. Area B marks a menu bar which consists of:

- Ontologies
- Ontology Design
- Semantic Mapping
- Background Information
- Community
- Login
- FAQ
- About

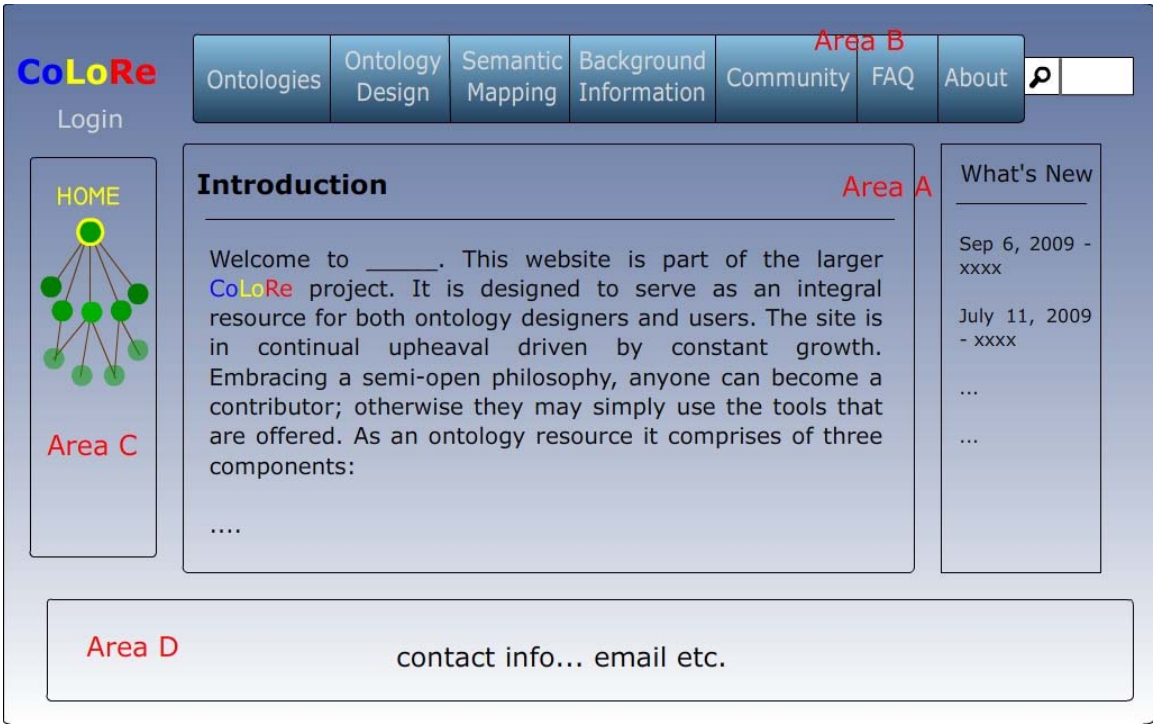


Figure 6-1 – Resource Main

Area C consists of a location tree, which shows where in the site the user is located in a sitemap type format, this will be clearer for pages located deeper in the site. Area D contains basic contact information at the bottom of the page.

6.2 Browsable Ontologies

As noted in chapter 2, one of the goals of a repository is to allow users to browse the constituent ontologies and to visualize their intra and interactions. While the relations between axiom classes are expressible as simple containment links, the axioms themselves are written in FOL and require more than a simple directory/tree browsing structure (c.f. BioPortal or OBO).

6.2.1 Ontologies

The ontologies for each of the core hierarchies are stored as a text file in CLIF notation. While this notation is the logical backbone, many automated theorem provers require

slightly different notation for reasoning; thus, depending on which Automated Theorem Prover is utilized, it may be necessary to duplicate the axioms in another form as well. Additionally, users should be able to browse and explore each of the axiom classes.

In general, the reason each hierarchy is referred to as a "core" is because interactions between them may not be known. Thus, each represent a different, isolated one to explore, though in some cases, mappings may exist (Groups – Fields (?)). Moreover, as noted earlier, the selection and inclusion of each core hierarchy lends itself to representation as a "map." It is suggested that this map be represented in two forms, one a zoomable, nested model space, as illustrated in Figure Figure 6-2, or as a tree-like structure as previously shown in Figure 3-5.

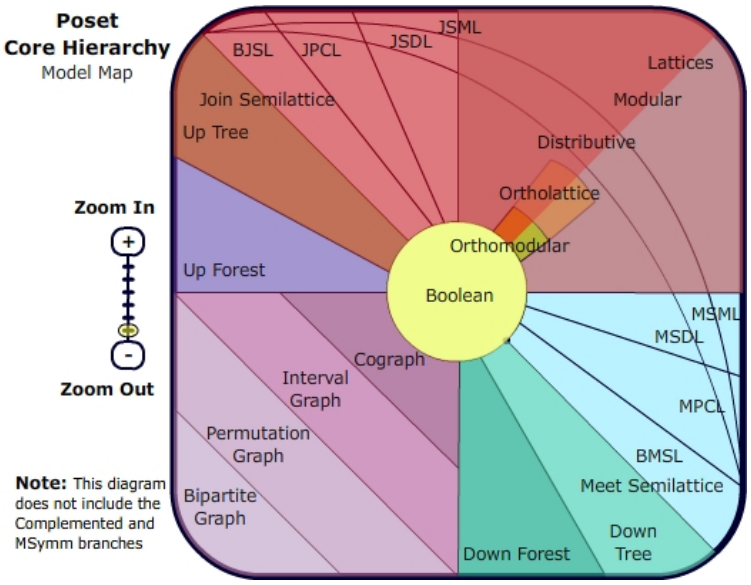


Figure 6-2 – Hierarchy as a Model Map

In either case, it seems natural to represent each core hierarchy horizontally across the page. Thus, the user may switch whether the tree or Venn diagram view is used to navigate. Clicking on a "(+)" beside any of the core hierarchies would expand its constituent elements according to the selected view. At this level of view, only concept names are shown for each class of axioms, while hovering over any of the names would show a preview of what is then accessible.

Clicking on the actual name of any of class of axioms would "zoom" the map in on selected class, expanding to the view as shown in Figure 6-3. The figure above has the tree view selected, where to the left, the relation of the class to the classes immediately above and below are shown (with classes connected but further away increasing in transparency). Clicking on any of these bubbles would zoom the navigation on the selected class of axioms. To the right, three elements are shown:

- English Description of Class
- Superclasses
- CLIF Axioms
- Models of Axioms

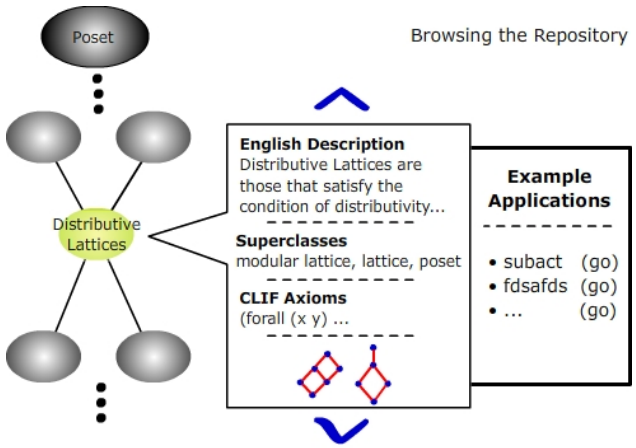


Figure 6-3 – Browse Hierarchy as Tree

The *English* description attempts as best as possible to communicate the "essence" of the axioms; namely, the restrictions that the axiom places on how objects of the relationship may "behave" with respect to one another. Essence and behave refer to the properties that the objects of the relation need to satisfy or exhibit.

The CLIF axioms by default only show the axioms which are associated with this extension. The user might note that above these listed axioms are a series of classes that connect the current one to the root axiom. Clicking on any of these would prepend their related CLIF axioms to the ones currently shown. For example, the current view shows

the axioms for a *distributive lattice*. Clicking on *modular lattices* from the list would prepend those associated axioms. Moreover, readers might also notice the *** symbol beside some lines of the CLIF. These indicate that the particular axiom may be reflected in the model provided below. Clicking on the symbol would highlight or initiate an animation of the relevant groups of objects as contained in the model which illustrate the restriction.

Models of the axioms are also included to illustrate their properties holistically. The default view is a paired down representation. In Figure 6-3 above, distributivity is highlighted in the model. An additional description regarding the highlighting is also visible just to the right of the model, which communicates the limitations of such a representation. The *Examples* component lists relations based either on semantic mappings or other ontology designers which have used this class of axioms. Finally, the up and down arrows located at the top and bottom of the box containing the above three elements expands a list of classes located above and below, allowing navigation in that way as well.

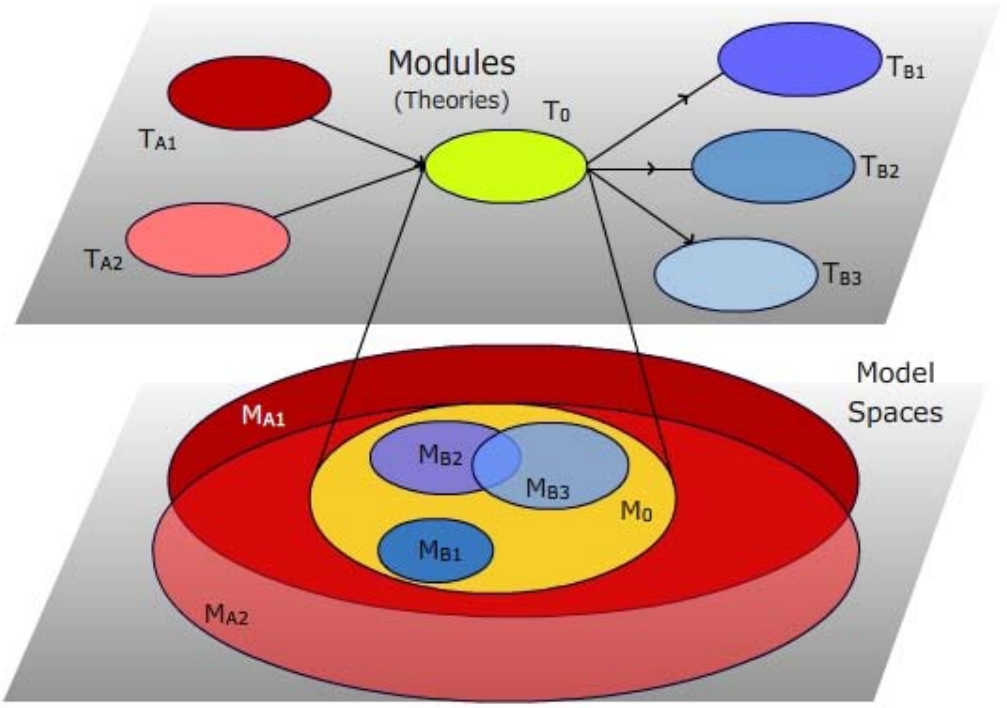


Figure 6-4 – Browse Hierarchy as Tree and Map

In contrast, the Model Space view shows the same tree structure, in a 3D format, with each class projecting onto a map of the models, as illustrated in Figure 6-4. The additional information, detailing the English descriptions, axioms and models contained to the right of the location within the ontology is the same as above. The advantage of this view is that it shows the relation between the models of the axioms and their connection to one another in tree format.

6.3 The Design Process (Algorithm Selection)

The algorithm and the list of examples are then accessible via the *Ontology Design Tool* link accessible via the menu bar. Once this link is clicked, users are directed to the *Ontology Design Tool* page, as illustrated in Figure 6-5. This diagram illustrates the default view, where the overarching navigation bar for the hierarchy is located at A. Area B contains a brief introduction to the ODT and contains links to *Learn More*, and *Get Started*.

Figure 6-5 – ODT Main

Learn More, transports users to several brief paragraphs which outline the philosophy of this approach as well as discussing some of the limitations inherent in the current implementation of each of the algorithms. A link to a FAQ for this section is also provided. The text for each of these may be found in Appendix B.

Get Started presents users with four options, *Search Applications*, *Interactive Navigation, Sandbox Mode* and *FOL Tutorial*. Each of the preceding options will be discussed in detail below.

6.3.1 List of Applications

Search Applications, as indicated by the brief description in the figure above, directs to the list of applications, which is a catalog of how each mathematical structure has been applied in particular domains. Users are provided with two options regarding their search, they may either browse through the class of mathematical structures, and see where they have been used, or they may instead browse the list of relation names from various domains and see which structures they correspond to.

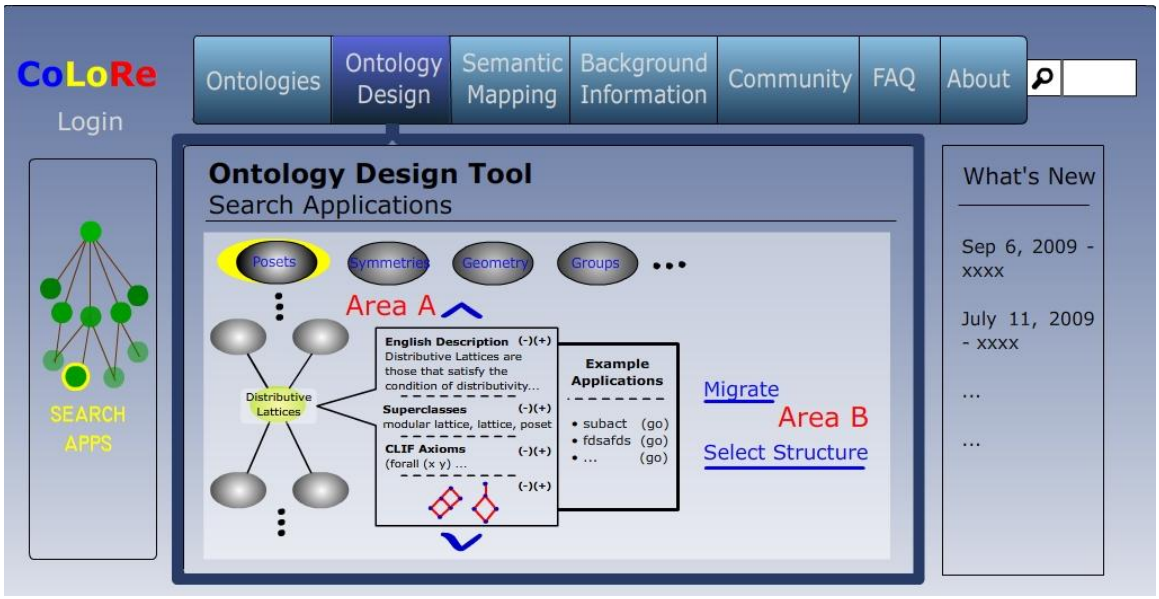


Figure 6-6 – ODT Search Applications

Figure 6-6, illustrates the view one would have, when searching for all domain applications where Distributed Lattices have been used. Area A is very similar to Figure 6-3 and includes lists the name of the class of structures currently being viewed, how the structure relates to its overarching hierarchy and contains the list of domain specific relations that are using the axioms for this class of structures. Furthermore, clicking on any of the domain specific relation names would also transport users to see if such a domain name has been used by others, in conjunction with other theories. Finally, Area B allows users to either "migrate" the current class of structure to begin the design process, allowing the user to refine the model to match their intuition, or if the current structure matches what the user desires, they may instead click *Select Structure*.

Figure 6-7 – Provide Information

If *Select Structure*, is chosen, then the user is first asked to provide some information about their relation, so that it may also be added to the catalog of examples, as illustrated in Figure 6-7. This page gathers the necessary information so that the axioms which have been written using mathematical terms may be automatically translated to the user's vocabulary. Thus, several types of information are required, namely:

1. Name the ***relation*** or ***function*** they are trying to define (i.e. *subactivity*).
2. Select whether the name refers to a relation or a function.

3. Enter the number of ***arguments*** the term consists of.
4. Define, if applicable, the ***sort types*** the relation applies to (i.e. *activities*, *atomic activities*).
5. Optionally, provide a ***URI*** for the source ontology where this term will appear in.
6. The last Field C provides users with an opportunity to input example names for the ***objects*** (i.e. Alice, Bob, Carol, Dave etc…). However this step is optional and is included only to allow models to be constructed using names that mean something to the ontology designer.
7. Select at which ***level of abstraction*** they would like to use the repository.
8. Select which ***core-hierarchies*** in that level of abstraction they would like to explore.

Once this information has been inputted, users may then click on *Verify*, where they are directed to the *Verification Step*, where models are presented to user to make sure the selected structure corresponds to their intuition. If users desire neither automatic translation of the mathematical axioms, nor for their relation to be added to the repository, they may simply click *Skip*, which transports users to the same page as *Verify*.

This component of the ontology design tool, applies to the following algorithms as well. At this stage, a model generator, such as MACE and an in-house developed graphical translator are used to generate the models that are to be presented to the user. Should the user select *Mismatch*, then they are given the following options:

- Learn More
- Begin Design Process
- Start Over

The first of these discusses the theory behind models and mathematical structures, and leads into the design process. *Begin Design Process* as the name suggests, initiates the algorithm and the interaction between the user and the software. *Start Over* simply redirects the user to the *Get Started* page described above.

On the other hand, if the user verifies that these models, are acceptable, then by clicking *Accept*, the CLIF axioms for the structure are presented, both using the initial mathematical language (as stored in the ontology repository), and another using the relation names and sort types provided by the user in the *Provide Info* panel described above. In such a way, it should be both clear to and easy for the user to integrate the provided axioms into his/her ontology.

As discussed earlier, the list of domain applications presents a straightforward search opportunity to a user, which involves exploring the catalog of relations to find a matching or similar relation. This feature of the site would attain greater relevance as the ontology repository grows; particularly so if it achieves critical mass. By linking previous defined relations to their originating ontologies, users may additionally discover an appropriate starting point for their design process, skipping a lot of elementary work that may be required in the navigation algorithm. This is especially true if no relation in the catalog is an exact match with their understanding of the relation. Consequently, users might choose to use a definition in the catalog as a starting point and then switch to the design environment.

6.3.2 Design Process

Selecting *Interactive Navigation* begins the ontology design tool algorithm. Users are first presented with the *Provide Info* module, detailed above, with a brief description for the implemented algorithm as shown in area A in the figure above. Once more, clicking on *Learn More* directs users to a more comprehensive description of the mechanics and theory behind the algorithm.

At any time, users may simply click on *Begin* to commence the process of axiom selection. If they have not provided the necessary information in the *Provide Info* module, they will be asked to verify that they wish to skip this step, and then begin the process.

The ontology design algorithm then provides users with the following view as illustrated in Figure 6-8. Here, users select which level of abstraction they wish to explore. Once they have selected the level of abstraction, they will further be able to refine their search limiting it to a subset of the core-hierarchies if they wish. The user may unselect a box, thereby removing that core hierarchy from the algorithm's search space. The default option is to search through all the core hierarchies. Additionally, users may select *Advanced Start*, if they wish to bypass the default initialization for the hierarchy, and instead wish to begin at some intermediate class within the hierarchy without first providing models.

Figure 6-8 – Select Hierarchies

Clicking *Advanced Start*, initiates a pop-up asking the user to select their desired core hierarchy, and then pops up a diagram of the taxonomy structure for those types of mathematical structure. The user then clicks on the class of structures they wish to initialize the algorithm at, and are then transported to the appropriate stage of the algorithm.

Otherwise, once user clicks on *Start* initiating the *Sandbox* environment, whereby users are asked to create at least one model of their relation. Figure 6-9 shows this software component with several user generated models.

6.3.3 Sandbox Environment

The final algorithm is situated in the sandbox environment, which allows users to graphically construct an implementation of the relationship they are trying to axiomatize. The current implementation is limited to relations or functions that may be visualized via a graph structure (virtually any binary relation). It uses a simple interface, where users populate the screen with nodes representing the elements their relation ranges over and edges the relation itself.

Figure 6-9, also shows an example where the user has already constructed three examples and has just finished the fourth. Area B, the history surface holds all the previous examples the user has constructed, and by clicking on any of them, that example is transported to the surface for further alteration.

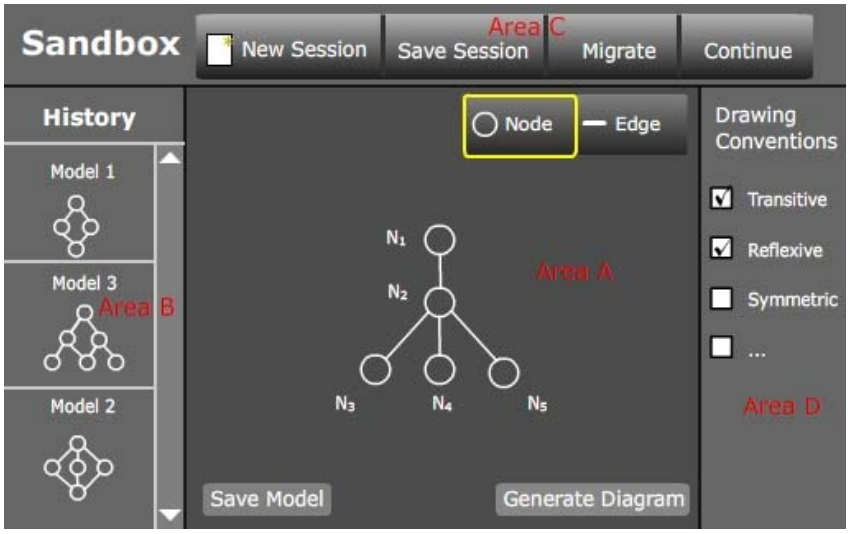


Figure 6-9 – Sandbox

As illustrated in Figure 6-9, one is presented with two surfaces and two menu bars. Area A is the main drawing surface, B the history surface, while area C corresponds to the

menu bar for populating the main drawing surface with examples. Clicking the "Node" button, allows users to click anywhere on the surface to insert a named node, while clicking the "Edge" button allows users to connect any two nodes. Area D consists of a series of check boxes to allow users to define the convention they would like to use in drawing models such as whether reflexivity holds without showing an edge going into the same element, or whether transitivity holds without requiring all those extras edges, etc.

When users are satisfied that their constructed diagram correctly captures a particular model of their relationship, they may select "Save Model," located in the main drawing panel. This option asks users to name their example and then transports the current example to the history surface as well. Using a name for a pre-existing example overwrites the current one to the saved one. Selecting "Save Session" would allow the user to save all generated models in the history for future reference. "New Session" would delete the history and start a new set of user models. "Migrate" would show which theories in the repostiroy the current model in the main panel corresponds to.

Selecting "Generate Diagram" would translate the diagrams into a series of FOL sentences, and engage the algorithm to derive the appropriate set of axioms that match the examples. There is no limit to the number of examples a user may submit, however as the number of nodes and examples increases, the search time for the algorithm increases significantly.

As an example, assume that the user drew the diagram shown in the main drawing window of Figure 6-9. There are five nodes, with an edge between nodes 1 and 2, with node 2 also being connected to each of 3, 4 and 5. The interface would then generate the following output as the complete diagram for the drawn model:

(exists (N1 N2 N3 N4 N5)

(and

(E N1 N1) (E N1 N2) (E N1 N3) (E N1 N4) (E N1 N5) (E N2 N2) (E N2 N3) (E N2 N4) (E N2 N5) –(E N2 N1) (E N3 N3) –(E N3 N1) –(E N3 N2)

–(E N3 N4) –(E N3 N5) (E N4 N4) –(E N4 N1) –(E N4 N2) –(E N4 N3) –
(E N4 N5) (E N5 N5) –(E N5 N1) –(E N5 N2) –(E N5 N3) –(E N5 N4)
)
)

Equation 6-1

These statements constitute the "world" of the diagram, and the most specific class of axioms that correspond to this user model would be found in each of the selected core hierarchies, if applicable. Thus, the model for which CLIF axioms were generated in Equation 6-1, matches the theory for *up-tree* in the poset core hierarchy. Finally, clicking "Continue" would transport the user to the second stage of the algorithm where the software proposes models for acceptance or rejection. It would first map each user model into the repository and having checked whether the algorithm applies, it would situate the user in the appropriate place(s) in one or more of the core hierarchies. The interactive navigation component of the ontology design tool would now be invoked.

6.3.4 Interactive Navigation

The first, second and third cases, lead us to the navigation phase of algorithm. Through the sandbox, users have now been situated at a single class of axioms in each applicable core hierarchy. Now, models generated by the algorithm are presented to the user in order to traverse to the most appropriate set of axioms given the repository. The fundamental component of all views in this environment is the core hierarchy, which the algorithm "navigates" through. As shown in Figure 6-11 below, the user is presented with a zoomed-in view of the portion of the hierarchy under investigation. Each structure is represented by the class name in a box, with the highlighted box being the one examined.

In the interactive navigation, this view consists of the following elements:

1. Visual Representation of Structure
2. Set of FOL axioms in CLIF
3. English descriptions of Structure
4. Migrate to Sandbox
5. Go to List of Examples for this Structure
6. Model Question and Feedback

6.3.4.3 Visual Representation

The key drivers of the algorithm are the visualized models for the theories. They take the centre stage in this view and the user must simply decide whether the presented model(s) are acceptable or not. Clicking "Accept" would put these models into the "Accepted Models" bin directly to the right. Conversely, "Reject" would add it to the corresponding bin. Either response would move the algorithm forward and closer to providing the user with the strongest theories in the repository that match his/her relation.

6.3.4.2 English Description of the Structure

Should the user click on "English," an English description of these class of axioms will pop-up and be made available to the user. Additionally, since models are not simply one for that class but may be constructed by negating a contained class, these nuances will be provided.

6.3.4.3 FOL Axioms in CLIF

By clicking the "CLIF" button, the axioms for these models may be viewed. Again, the axioms will change depending on which stage of the traversal the algorithm is at, since the models correspond to various negations and unions of the selected theory and those above and below it.

6.3.4.4 Migrate to Sandbox

This component of the dialogue box for each class allows users to switch to the S*andbox* environment, by exporting the appropriate diagram for the structure. In the current implementation, it exports the Hasse Diagram that is associated with the selected class. In

this way, if users have found a diagram that is similar to what their concept implementation might produce, but differs slightly, they may export the diagram to the Sandbox environment and adjust it accordingly.

6.3.4.5 Go to List of Examples for this Structure

Similar to the *Migrate to Sandbox* option above, this allows users to locate examples by other ontology designers that utilize the selected class of axioms. By clicking this button, users are then directed to that section of the repository.

6.3.4.6 Model Question and Feedback

As discussed in the algorithm section, the model space of a particular class of axioms is partitioned according to the classes connected immediately above and below it. The two options for this question are "Accept" and "Reject" corresponding to whether the presented model is acceptable to the user or not.

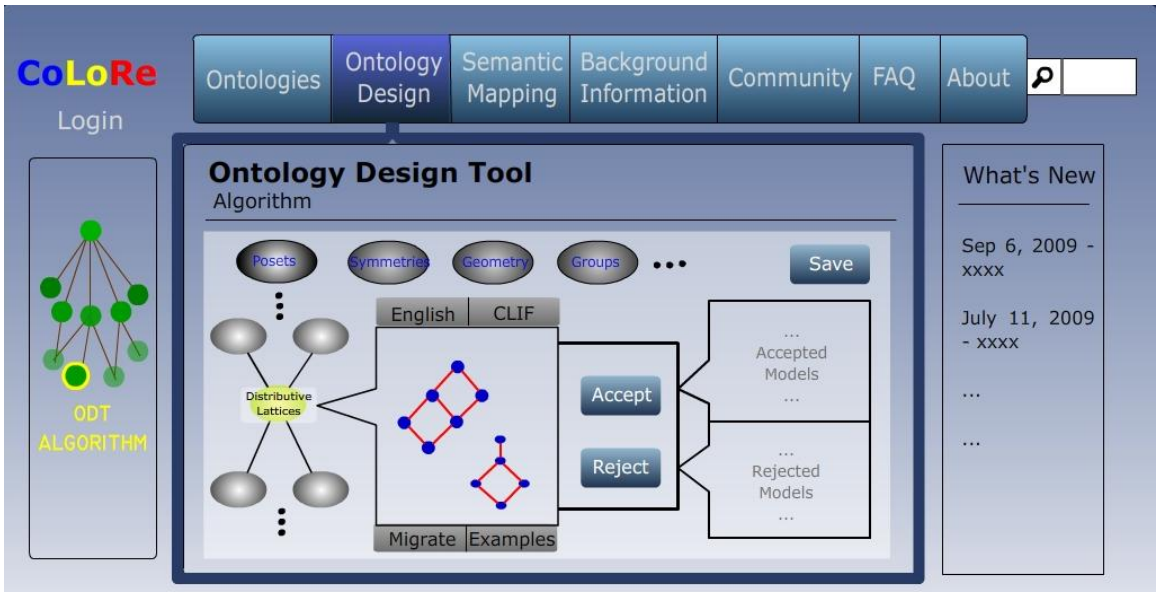


Figure 6-10 – ODT Engage

The user responses drive the algorithm along the map until there are no more models to present to the user. As shown in Figure 6-10, the user is shown which theory they are currently investigating as well as all Accepted and Rejected models. Once all interesting models have been vetted and all other core-hierarchies have also been traversed the

algorithm would proceed to present the strongest axioms in the repository for the user's relation. If the user provided a relation name, then the axioms will be presented in CLIF using the user's relation, otherwise it will be presented using the relation names as they appear in the repository.

6.4 Semantic Mapping

The other important module of this service is the semantic mapping component. Users may reach it by clicking on the Semantic Mapping link available on the navigation bar, or on certain pages. Doing so, presents the user with the screen shown in Figure 6-11. The interaction here is quite simple, as the options are limited to mapping a target ontology to the repository, or using the repository to find commonalities among any number of ontologies.

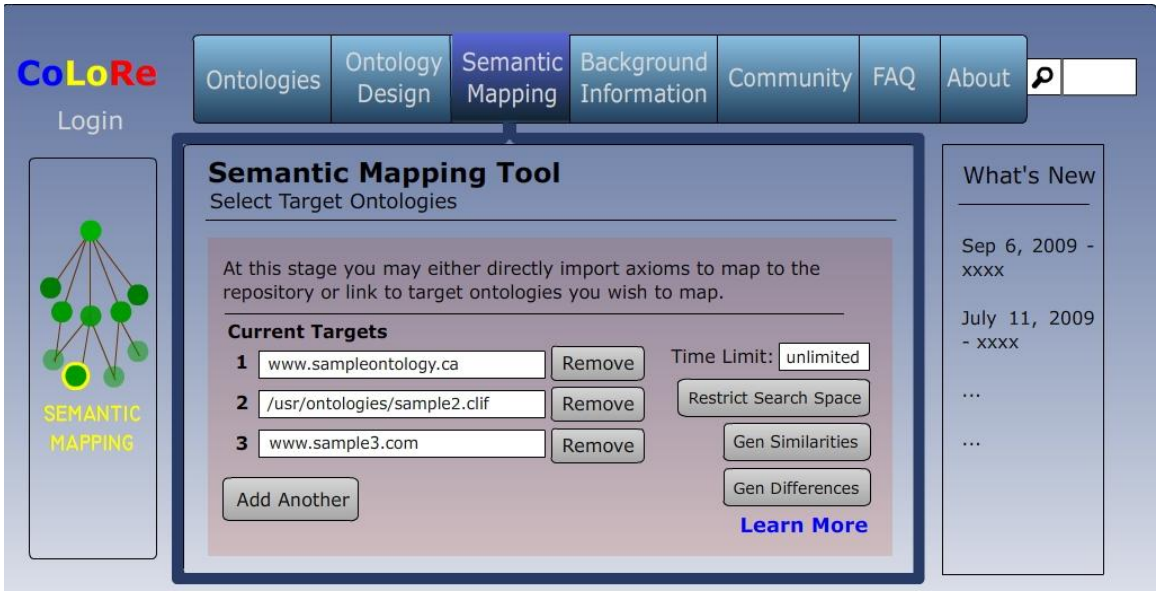


Figure 6-11 – SM – Select Target Ontologies

As before, a *Learn More* link is available to allow users to explore the underlying assumptions and mechanisms of the mapping. To engage the mapping, users may either point to a URL, or upload a text file containing the axioms to be investigated. It should be noted that the target axioms at the current stage, are required to be in CLIF form.

Support for translation from other common ontology languages to CLIF is a desirable future addition. Users may add further ontologies for comparison in the same way.

Optionally, as shown in Area C, a time limit may also be set to cut short the mapping if it is taking too long. The user may also click the "Restrict Search Space" button to limit the semantic mapping search to a subset of the repository by checking or un-checking boxes corresponding to each of the core hierarchies. With all of these options selected and accounted for, the user may click on "Gen Similarities" or "Gen Differences" to conduct the mapping. When clicking either, a pop-up would ask the use to define ONT* for the similarity construction and which ontology to single out for the differences.

	Ontology 1 (O_1)	Ontology 2 (O_2)	Ontology 3 (O_3)
Relation Names	Part_Of	Flows	Subactivity
Repository Match	T_1, T_2, T_3	T_1, T_2, T_5, T_8	T_1, T_2, T_5, T_6, T_9

ONT*	Similarity	Diff (O_1)	Diff (O_2)	Diff (O_3)
ONT*1 = {O_1, O_2, O_3}	T_1, T_2	T_3	T_5, T_8	T_5, T_6, T_9
ONT*2 = {O_2, O_3}	T_1, T_2, T_5	N/A	T_8	T_6, T_9

Figure 6-12 – SM Output

As discussed in the semantic mapping chapter, the algorithm conducts a breadth-first specialization search for each ontology independently for each selected core hierarchy. Each ontology would have its own column, with the name of the relation or function being mapped into the repository displayed. The next row would show the image of the ontology in the repository, consisting of all theories in the repository that were consistent with the ontology. Another table would be for each selected ONT* and the Similarity and Difference measures for each, as illustrated in Table 6-1 above.

6.5 Additional Features

These are elements of the site that are integral to its function but ancillary to the repository and algorithms. As noted above, users must be able to access information concerning the purpose of the site, authorship, background and links to information about the repository community.

6.5.1 Background Information

As has been stressed throughout this work, users of this repository need not be particularly familiar with the intricacies of first order logic. However, being able to converse or at least understand the basics of this language is a secondary goal. To this end, users should be provided with adequate material to be able to appropriately understand the possibilities and limits of what is presented here. Consequently, the "Background Information" link provides users with such a necessary grounding.

It is debatable whether it is beneficial to reproduce such explanations from scratch here, or whether to instead link to other sources which do much of the same job. One issue here is that the repository focuses on elements of FOL or the snippets of logic as pertaining to the core hierarchies as they relate to ontologies as knowledge representation. Many external sources provide a more general description of the same concepts, denuded of the particular context as desired here. Nonetheless, in the spirit of reusability and extensibility it seems natural to capitalize on work already done instead of reinventing the wheel.

Additionally, one notes that logic related articles on sites that use GNU Free Documentation License (i.e. Wikipedia) are generally of high quality. Consequently, a two pronged approach is suggested for this section of the website. Where possible, links (or framed versions, this decision is left to the web designer) are provided to the user to explain the overarching principles of ontologies, logic etc. Extending this information,

following each of these basic sections, is a section developed here that shows how these issues relate to ontologies in the repository.

Here, the user is presented with a brief description of FOL, an expandable link to the wiki documentation regarding this language, as well as an additional block of text linking the language to the ontologies in the repository. Readers might note that this text is similar to that provided in the literature review and background chapter of this thesis. The main areas covered in the background information section are:

- Ontology
- Logic
- Semantic Mapping
- Further Reading

We note again that this is a secondary (tertiary?) goal of this thesis, so there is much more work that can be done here. Additionally, there does not exist an easily accessible resource that documents papers, articles or discussions pertaining to many of the issues that are explored by this thesis. The further reading component of the background information strives to collate many of these sources in one place, allowing the interested user relatively straightforward access to these documents.

6.5.2 Community

The community section of the site consists of forums where issues regarding the site can be discussed. This section is currently sparse, and depends on the interest that the ontology community shows in this effort. However, the basic infrastructure to support and facilitate participation in this project is important. To that end a web forum is provided to allow users to propose additions that they might like to see, or whatever issues might be relevant to the site.

Such a forum is a venue to reconsider changes to the guidelines of the repository, interact with other users regarding implementations of the contained ontologies and so on. Users may also request to join as one of the overseeing contributors who verify changes and novel ontologies. Users may also subscribe and receive emails from discussions from the forums or whenever an alteration is made to the repository.

6.5.3 Login

The login link is a very straightforward component of the site. It leads to the account system, allowing users to save sessions in an algorithm or to make changes to the repository. The particular implementation choices of such an account system are beyond the scope of this thesis.

6.5.4 FAQ

This page lists some of the anticipated common questions regarding the repository and the algorithms. At this point in time it is unclear what these might be, so no content has been generated for this section. However, one would certainly expect to find this component in the final software artifact.

6.5.5 About

This section contains information regarding authorship, the philosophy of this service. Authorship lists contact information for the contributors and maintainers of the website. Philosophy of the service lays out at a very high level the purpose of the repository, what it hopes to achieve and how it aims to integrate into the ontology community.

Chapter 7

Recap and Future Work

7.1 Recap

This section will revisit the concepts that provided the impetus for this thesis, and the degree to which each identified problem was addressed. The first problem arises in constructing first order formulations: specifically, axiom generation poses a significant hurdle for those not formally trained in the corresponding logics. Often, it may feel unnatural or unintuitive to express an idea in the syntax and grammar of FOL. The second problem arises in trying to understand how exactly one might use two or more ontologies together – particularly, how might one generate mappings to enable inter-use between ontologies.

In providing solutions to each of these two problems, this thesis essentially extended the role metaphor plays in the reuse of thought structured by languages. It should be made clear that no claims have been made as to how the human mind actually works or forms thoughts. Rather, it is simply asserted that it is useful to generate ontologies on the assumption that whenever one uses a particular logical language, there will be snippets –

pieces of logic lego – that underlie the more complex structures that correspond to abstract phenomena that we choose to name or describe. Since any language, especially a formal language, biases what can be stated, the underlying logical structures are often reused. Indeed, as shown in **Error! Reference source not found.**, as this basic scaffolding is being constructed, one can simultaneously affix more abstract concepts on top, adding increasingly complex layers of abstraction and understanding. We hope to capture this process in a semi-open repository that grows, in part, in accordance to the insights of theoreticians and at the same time in response to the needs of practitioners.

7.1.1 The Ontology Repository

The language of FOL, and in particular Common Logic, has associated with it many classes of theories noted as particularly useful or interesting by mathematicians and computer scientists. While these logical constructs are not the only ones of relevance or utility, they do represent classes of theories that are right now, well understood and characterized. Moreover, as these classes of theories are very basic observations about types of consistent and useful logical patterns constructed in the language, they suggest themselves as particularly good candidates for the lowest level of the ontology. The realization of these theories as "logic lego blocks," leads to the creation of the ontology repository that is the engine into which the two modules plug into. To use another metaphor, each hierarchy in each layer of the repository corresponds roughly to header files one might use in programming. The current layer of mathematical theories is similar to the "math.h" library header file that is included in many C programs.

7.1.2 Ontology Design Aid

The axiom generation module exploits the relationship between the axioms and models of an ontology to elicit what an agent intends via a simple communication process. The same idea of meaning, captured in formal logics as models, may be translated and represented by a variety of syntactical formulations. Whereas most people who lack formal training in FOL or CL have difficulty articulating their thoughts in the syntax of

the language, it is conjectured that they would have less difficulty identifying a possible manifestation of their meaning in the guise of a model.

Moreover, capitalizing on the inherent taxonomy (or hierarchy of extension) that is arises from the non-conservative extensions of theories in each core hierarchy, we are provided with a map to navigate. By simply requiring users to generate at least one example model which is then used to generate a complete diagram, the first part of the algorithm situates users in our "map."

By noting the relation between models of axioms and the intended models of the user we then draw on the scientific method for inspiration. Since the set of models for an ontology is in general, infinite, we can never be certain that the axioms we are proposing are exactly what the user wants. However, should the user reject a model that is associated with a particular theory, we may with certainty reject that theory since it has been falsified. This process allows us to use the same representation (usually graphical) that the user first employed in creating their model to present these falsification models. Rejecting or accepting a model propels the user through the hierarchies, resulting in the *strongest* (as defined earlier in the paper) set of axioms found in our repository that is consistent with the models the user has accepted and rejected. In many ways, what has been developed here is a machine learning algorithm, where the intuition of a human designer is slowly refined and learned by a simple feedback process.

7.1.3 Semantic Mappings

The problem of semantic mapping is also turned on its head. While most of the ontology community has been transfixed with description logics and has hit a rabbit hole in trying to generate (and automate) mappings, the process developed here can go straight to the semantics. In this case, it doesn't matter whether an ontology designer has labeled a relation "larger" or "bigger," it is the axioms which are important, not the names. To quote Richard Feynman:

> *You can know the name of a bird in all the languages of the world, but when you're finished, you'll know absolutely nothing whatever about the bird... So let's look at the bird and see what it's doing — that's what counts. I learned very early the difference between knowing the name of something and knowing something.* (Feynman 1969)

We first note that the mappings developed in chapter 5 are limited to only those theories which are represented in the repository. Everything is understood through the prism of what is represented in the repository. With that caveat aside, generating mappings based on consistency becomes relatively straightforward.

Any number of target ontologies can be mapped into one another, relative to our repository. First, an image of each ontology is constructed given the theories present in the repository. Next, a global intersection among each of the images is taken, resulting in the strongest set of theories found in the repository that is consistent with all the target ontologies. Moreover, the differences may also be explicated (pair-wise) for reference.

While the algorithms for the two modules and the guidelines for the repository have been made explicit, full-blown implementation is beyond the scope of this thesis. Thus far, what will have been implemented will be the ontology of partially ordered sets in CLIF and the design conceptualization for how the repository and modules should take form. Of course, the correctness proof for the algorithms, and a proof of concept the architecture of the software artifact via the use cases has also been provided. The next section will detail how the ideas presented in this thesis may extended.

7.2 Future Work

The three elements presented throughout this work are only the starting point for an exciting program to allow us to realize the full potential of ontologies. Extensions of each component will be provided one by one in the following paragraphs.

7.2.1 The Ontology Repository

There are three areas where further work may be undertaken on the current formulation of the repository. The first is related to the ontologies which comprise the repository, while the second raises interesting questions about the implications of such a repository for ontology architecture as a whole. Finally, the third point addresses issues arising from its manifestation as a web service, consisting of a community and guidelines for its management and maintenance.

7.2.1.1 Extending the Repository

Thus far, this thesis has focused on cataloging FOL expressible theories that have been noted by mathematicians and computer scientists. It was briefly mentioned in chapter 3 that one possible extension of the repository is to allow bottom-up theories to be identified and added to each of the hierarchy maps. For example, a user may want a tree where each parent has exactly three children. While this theory is currently not in the repository, guidelines can be developed for how to incorporate these "novel" classes into the hierarchies (based on frequency? or some other criteria?). Expanding this thought, the only real criteria as to what constitutes a potential candidate ontology for the repository is that its models must be well understood.

The only reason that the first iteration of the repository is limited to mathematical structures is that they are the most well known. Moreover, they represent the primitive concepts for a formal language such as FOL. More complicated theories, such as those for space, time mereology, topology, events etc. would constitute the following level. Further complicated theories about say geographies, organizations etc. would naturally be represented above these. As the implications of the models of these theories become explicit, characterized and better understood, these ontologies would work very well in the repository.

To reiterate, the repository in its current form represents a "math.h" library. As is well known, there are many other useful functions that collected by other headers. There is no reason to simply limit candidate ontologies to only mathematical structures. It is important to also contrast this approach to that of the traditional ULO's. Most ULO's make a claim along the lines of "*this* is what time is" to the exclusion of alternative representations. A repository such as the one developed here allows people to "plug-in" to the most appropriate set of time axioms as per their use. Moreover, given that each set of theories is also organized taxonomically via non-conservative extensions, it greatly facilitates the questions of how ontologies using different representations may communicate with one another.

7.2.1.2 Upper Ontologies

Furthermore, it is conjectured that much of the problems plaguing Upper Ontologies can be mitigated by paying closer attention to developments in cognitive linguistics and cognitive neuroscience. While I have not explicitly invoked work in either of those fields, nor would I suggest that the ontology community take a position on how the human brain works, we should pay closer attention and capitalize on theories developed by these communities to inform our pragmatic and problem solving oriented artifacts.

Each Upper Ontology carries with it a series of implicit biases as to how the world really is. The effects of these biases are further propagated implicitly to whatever other ontology plugs into it. Heeding the words of McLuhan, where "the medium is the message," the repository outlined above instead makes explicit the biases of the formal *language* we are using. The structures that are identified as those that are useful to describe things in the world as seen through formal logic.

These are the structural "biases" which affect everything else. It is only when these have been made explicit, that the bias of cognitive or philosophical perspective comes into play, whereby we match phenomena in the world to these structures. An area of

interesting topic for research would be to use a "meta-Upper Ontology" to make explicit the assumptions of the actual Upper Ontologies. For example, in formulating DOLCE, Guarino et al explicitly state (though not in any formal language) that their choice of fundamental terms if motivated by research in cognitive science. Other ULO's have different biases that are not articulated in a formal language either.

7.2.1.3 Relations between Modules

As was noted in chapter 3, the relationship between modules is currently represented externally to the axioms. While the (cl:import()) term implies that one module is reusing axioms from another, the exact nature of how the importation is affecting the theories one can construct based on the current module is ambiguous. Moreover, equivalence between modules, or the fact that one module might be the extension of not just a module it imported but another, is also currently not represented.

One option would be to use the novel expressiveness afforded by CLIF to range over quantifiers, and in this way, make (if) and (iff) statements corresponding to extensions and equivalence respectively. Another idea might be to define a mirror repository which captures the relations between the ontologies. For example, we might define binary relations for "equivalence," (Equiv X Y), "conservative extension" (ConsEx X Y) and "non-conservative extension," (NoConsEx X Y) which holds true for X and Y. Of course, this is a removed level of abstraction as we cannot directly connect this repository to the repository of the actual theories otherwise we would be moving into higher order logics. One would need to already know the relationship between theories X and Y before putting them into the repository. Regardless of the avenue pursued to explicate the relationships between modules, it is a highly desirable project.

7.2.1.4 Extending the Repository as a Web Service

Another obvious area for further work is to implement the repository as a fully functioning web service. While the programming component of this endeavor is straightforward, as noted in previous chapters, it is important to develop clear guidelines

for its maintenance. Recently, the idea of an open-ontology repository has spurred some discussion on this front at the ontolog forums (Ontolog 2008). The transference of the findings there to the project here is unclear, nor has any consensus yet developed; nonetheless it shows that these issues are gaining wider currency in the ontology community.

Usually, a design process is iterative, and is especially dependent on feedback from potential users. Thus the repository is not even in *alpha* versioning, and would need extensive work to reach a *beta* stage, let alone a full fledged release. A warning to potential developers is that many such artifacts are designed with too little emphasis on the end-user experience, and too much on achieving the functional specifications. As one the objectives of this project is to make ontology generation and mapping accessible to those without formal training in logic, and particularly non-engineers, the interface for such a service needs to be welcoming, with a relatively low learning curve.

7.2.1.5 Scaling, Growing the Repository

There are two notable issues that arise in growing the core-hierarchies and adding new layers. We recall a stipulation that every theory in each core-hierarchy must not share a non-conservative extension with any theory in any other core-hierarchy. As new theories are added, care must be taken to ensure that this criterion is preserved. While this care ensures that there is a baseline in the understanding of interactions between these silos. Investigating whether there exist generalizable types of interactions between theories extended in distinct core-hierarchies is an interesting question.

Finally, empirical or further analytical exploration of the scalability of the theories and the tractability of automated theorem proving over large ontologies given current FOL and computing tools is also necessary. While both algorithms rely on a specialization approach, some difficulty may be encountered if the models drawn by the user are particularly complex, or a core hierarchy becomes very large. Moreover, while the fundamentals and correctness of the algorithms have been outlined, they would likely

derive much benefit from closer optimization with each core-hierarchy they might traverse.

7.2.2 Ontology Design Aid

The basis of this module is robust and generalizable, producing a novel method of eliciting axioms from subject matter experts without requiring them to be especially familiar with FOL. Any relation or function that can have models represented in an unambiguously translatable representation is a candidate for the developed algorithm. The current work has illustrated how this algorithm may be applied to binary relations or functions, and in particular using a node-edge graphical form. Identifying and building programs that allow a user to generate models (and then to automatic translate it to CL sentences) for various other representations is an area of further work. Again, we would reiterate that these representations need not be graphical, or even visual, they simply need to satisfy the following criteria:

- allow user to generate models (relatively) easily
- generate unambiguous complete diagrams for the models
- be able to represent automatically generated models in the same form

7.2.2.1 Infinite Models

We should recall that it is not always clear how to represent or "draw" models of infinite size. Examples might involve the use of an ellipsis, or perhaps the drawing can be an interactive environment, where the agent may expand (zoom?) and explore indefinitely until it is satisfied it has seen "enough" of the model. Given that we require a complete diagram to be drawn from a model, it would be necessary to define different types of infinite sequences. Thus, instead of using reasoners to check for the diagram against theories for consistency, the finite parts of the model coupled with the infinite sequence

would be recognized as patterns of the models, in comparison to a table in the repository, corresponding to theories in the repository.

7.2.2.2 Multi-layered Models

An interesting and useful corollary is that as domain specific ontologies become linked to those at different layers of abstraction (i.e. at the molecular-bio to mathematical-logical), then different visualization abstractions and constructs might intersect fruitfully. Consequently, interaction at the molecular biology ontology layer might be conducted using graphics corresponding to actual molecules. Thus users might discriminate using models of molecules that exploit the fact that these abstract symbols in the repository relate to accessible (often concrete) things. This is similar to how users can customize skins for software graphical user interfaces, model representations may be skinned, so as to better reflect a representation more natural to a SME's domain.

7.2.2.3 Mapping to Multiple Hierarchies

There are a number of places where the algorithm currently simply terminates without being able to help the user. First, if two subsets of the models generated by a user, M^1 and M^2 map disjointly into more than one core hierarchy, the algorithm does not apply. Were the mappings *between* each of these core-hierarchies better defined, the scope of the algorithm would increase, as this eventually could lead to an unambiguous result. Consequently, developing tools which address this contingency would be of particular use.

Similarly, if all user models map into the same hierarchy but different particular theories, and if the non-conservative extension of that theory is inconsistent, the algorithm currently terminates without a result. This situation entails a termination because the algorithm cannot distinguish among the possible cases leading to this inconsistency. If a "path" to traverse the map of theories in a hierarchy – perhaps by defining a "location" in the hierarchy, essentially bifurcation (or more) for the definition of the relation/function – this problem might be mitigated, and again the scope of the algorithm would expand.

7.2.2.4 Transcribing to Other Syntaxes

Finally, in terms of implementing this module, it would be useful to have programs which automatically translate CLIF syntax to whatever syntax the user may be using or most familiar with. Additionally, the ability to convert axioms to the syntax of a variety of theorem provers would also be useful. On that note, further research should be conducted as to which of the current crop of theorem provers is best suited to this application. This thesis would currently place Prover9/Mace at the top of the list, because of its accessibility (freely available) alongside the inclusion of an automated model generator. Integrating these components seamlessly into the overall web architecture presents another interesting challenge.

7.2.2.5 Automating the Discriminating Agent

Another area that would be very exciting to explore is the automation of the agent that currently inputs the "user" models, and also decides whether a model generated by the repository is acceptable or not. Extending the idea developed in the previous chapters in this way would likely require one to develop theories for different types of input or sensory data. More immediately, one might envision a scenario where agents imply have access to a large body of "experience," corresponding to data or knowledge bases consisting of assigned models of the world – i.e. multiple extensions of possible relations and functions.

A more desirable outcome would be to have one or more agents eliciting information from a variety of sensing mechanisms. Sensed phenomena (according to the sensor type), would be used to drive the construction of a "coherent" model of the world using salient features noticed in the incoming data stream. The salient features would correspond to different types of relations or functions as stored in the repository. Additionally, there would be a theory or likelihood function for how to assign relation to the patterns (leading to the construction of models by creating extensions) noticed in different types of sensory data streams. Hence, relations that cover visual phenomena might be more

readily used to parse a video feed. Similarly, as noted above, it would be necessary to define a theory of "coherency" for each sensor type – i.e. what types of patterns in any particular data-stream might one consider coherent/worth noting?

7.2.3 Semantic Mappings

The semantic mapping module presents an opportunity for further research as well. While its utility grows as the ontology repository grows (in an "S" fashion), what can be done with the outputs of the process are of particular interest. In general, a user is presented with two types of lists; one being the theories found in the repository that are consistent across all the target ontologies (similarity), and the other being those that are different (difference).

While the first lists elucidate the way in which relations from one ontology may be reused in the other, the second provides a variety of opportunities to contrast and see which theories in the repository make the relations inconsistent. In analyzing the differences among ontologies, it is possible to group the theories that entail inconsistency together in clusters. Different combinations of these clusters may exist for the same set of differences. Some of these groupings of the difference might suggest precise ways in which the application of the relations in the compared ontologies differ.

Moreover, by paying close attention to which theories are found in one and not the other, it may be possible to say which ontology is *stronger* (as defined in previous chapters) than another. As we noted earlier, we can currently determine whether one ontology entails another, is simply consistent with it, or is disjoint from it. Whether we can increase the strength of these results to cover complete equivalency is also of interest. If representation theorems have been proven for various classes of theories, and if for two distinct theories in different core-hierarchies, these representation theorems are isomorphic, then we would then also be able to confirm equivalence. Particularly, if the consistency between significant portions of target ontologies is established, and representation theorems exists for those theories in the repository, the results of the

semantic mappings might suggest a representation theorem for the target ontologies. While the proving of these theorems is not automated, the tool may aid knowledge engineers in identifying and pursuing representation theorems for ontologies.

This leads us to questions of the (informal) tractability of the algorithm. For our purposes we can utilize theorem provers in the following three ways: (1) show that A and B are inconsistent; (2) show that A implies B (or vice versa); (3) show that the complete diagram for a model *M* satisfies axioms for A and B respectively. A positive from any of these three propels the semantic mapping algorithm. For (3) we might generate and store models for each module, thereby only testing to see if the complete diagram for model *M* satisfies the target ontology as a check of consistency. Alternatively, we could work backwards, using models from the target ontologies and testing them against theories in the repository. All three approaches together would greatly alleviate the computational burden of executing this algorithm.

A further area of research is the extension of the semantic mapping algorithm to consider multiple relations simultaneously. As currently elaborated in this thesis, the algorithm makes use of a single mapping axiom to generate images in the repository and to consequently link the target ontologies. Combining multiple relations together and exploring the resultant interactions between them poses an interesting problem which requires significant further work.

7.3 Conclusion

Overall, this thesis developed a repository to collect the various logical substructures that are the biases of Common Logic. The vision is that these ontologies will constitute the basic scaffolding to allow more complex theories be built upon them. In addition to that role, these ontologies further elucidate the assumptions that underlie the particular conceptualizations and naming of phenomena, connecting them to their basic logic blocks. As noted above, instead of spending time and resources on identifying semantic mappings because of ambiguous semantics, by quickly establishing where target theories

are similar and different, the focus is now shifted to determining what interesting things can be said based on these mappings.

Finally, the ontology design aid reduces the hindrance of learning a new language (let alone being comfortable enough in one to formulate and articulate complex thoughts!). Instead, drawing on the scientific method via falsifying hypotheses, we have developed an algorithm in the style of machine learning, where a human agent communicates with a software agent, in essence "teaching" it what it means when it uses a particular term.

All these components comprise only the beginning of hopefully a larger movement to the use of more expressive (and ultimately more useful) ontologies. Regardless, it is hoped that the ontology design aid module will allow a greater number of subject matter experts to articulate what they know in a formal logic. The semantic mapping module addresses how to link various ontologies to one another, while the repository realized as a web service provides a focal point for those interested in formal ontologies to contribute, modify and in general engage in such a process. As noted above, much work needs to be done for the ideas presented here to realize their potential, and it is hoped that they are attractive and interesting enough to enough people that it *will* be realized.

Appendix A

Axioms for the Repository

Partially ordered sets are a class of binary relations that order elements of a set. They are often defined using the binary relation ≤ or ≥. For the purpose of this thesis, we use "lte" which represents less-than-or-equal-to. Greater-than-or-equal-to is simply the dual of lte. The order of the axioms presented in this appendix will roughly mirror that of the poset-hierarchy. There are some modules which are used to define basic terms that are reused in each of the poset classes, those modules are presented at the end of the appendix.

A-1 Poset Core Hierarchy

The poset core hierarchy is reproduced here once more in two forms. Figure A-1 below shows how the models of each module are contained in one another. Figure A-2 is a reproduction of hierarchy using lines to indicate which theories are non-conservative extensions of one another. Incidentally, the resultant structure is a join semilattice.

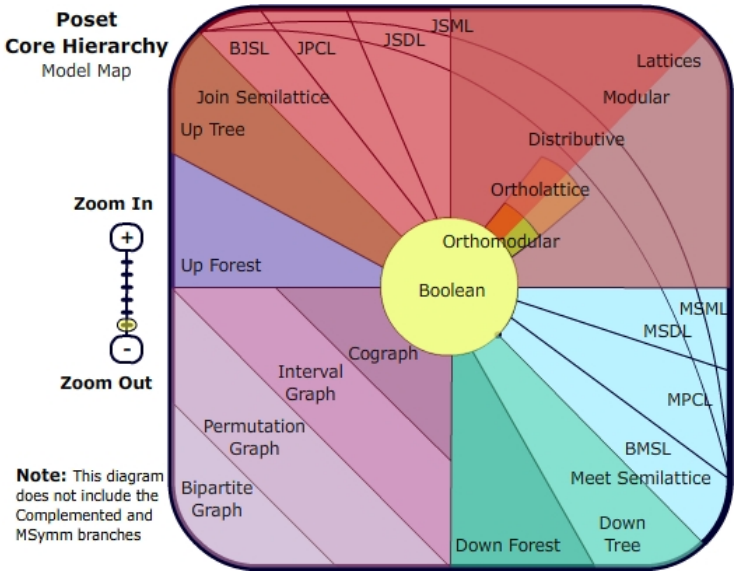

Poset
Core Hierarchy
Model Map
Zoom In
Zoom Out
Note: This diagram does not include the Complemented and MSymm branches
BJSL
JPCL
JSML
JSDL
Lattices
Join Semilattice
Modular
Up Tree
Distributive
Ortholattice
Orthomodular
Up Forest
Boolean
MSML
MSDL
Cograph
Interval Graph
MPCL
Permutation Graph
BMSL
Meet Semilattice
Bipartite Graph
Down Tree
Down Forest

Figure A-1

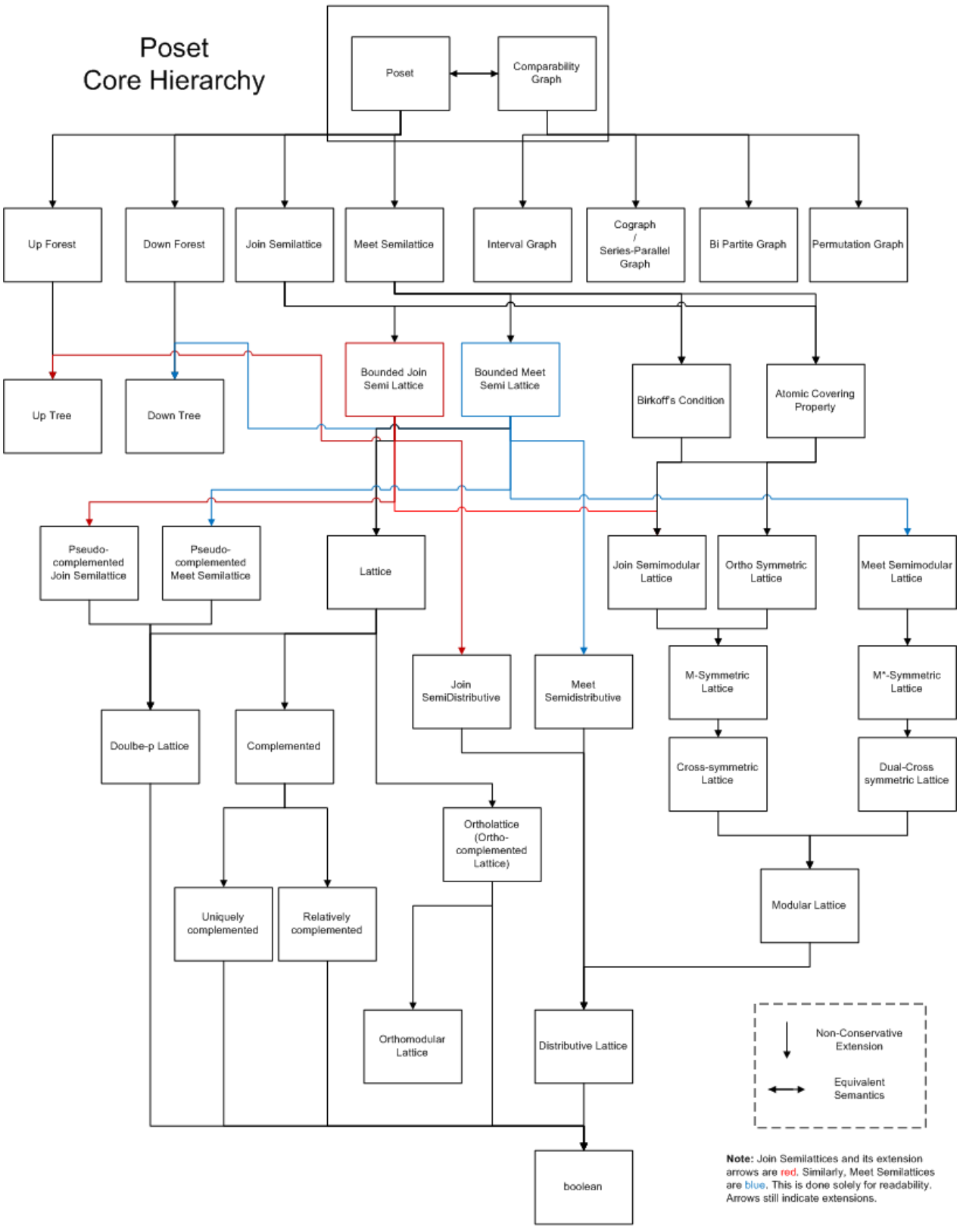

Poset
Core Hierarchy
Poset
Comparability Graph
Up Forest
Down Forest
Join Semilattice
Meet Semilattice
Interval Graph
Cograph / Series-Parallel Graph
Bi Partite Graph
Permutation Graph
Up Tree
Down Tree
Bounded Join Semi Lattice
Bounded Meet Semi Lattice
Birkoff's Condition
Atomic Covering Property
Pseudo-complemented Join Semilattice
Pseudo-complemented Meet Semilattice
Lattice
Join Semimodular Lattice
Ortho Symmetric Lattice
Meet Semimodular Lattice
Join SemiDistributive
Meet Semidistributive
M-Symmetric Lattice
M*-Symmetric Lattice
Doulbe-p Lattice
Complemented
Cross-symmetric Lattice
Dual-Cross symmetric Lattice
Ortholattice (Ortho-complemented Lattice)
Modular Lattice
Uniquely complemented
Relatively complemented
Orthomodular Lattice
Distributive Lattice
Non-Conservative Extension
Equivalent Semantics
boolean
Note: Join Semilattices and its extension arrows are red. Similarly, Meet Semilattices are blue. This is done solely for readability. Arrows still indicate extensions.

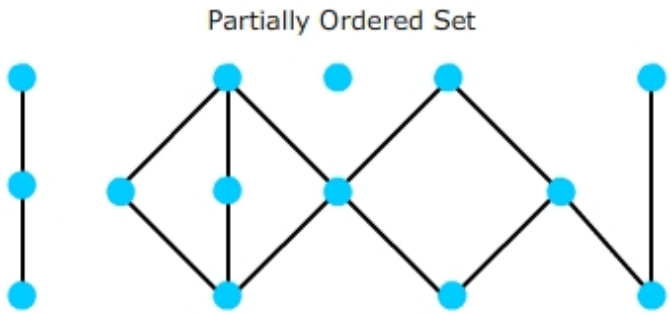
Figure A-2

A-1.1 Poset Axioms

A partially ordered set is usually defined via three axioms:

- transitivity
- anti-symmetry
- reflexivity

Occasionally, reflexivity is not included, resulting in a "strict" partial ordering. Below, a model for poset is provided, followed by the axioms in CLIF notation, with an English description and mathematical logic notation preceding the axioms in the repository as comments.



Note: This poset consists of a chain that is not connected to the rest of the elements. Moreover, it has an antichain located in the middle. Hence not all elements are comparable, nor connected.

(cl:module (poset)

(cl:comment "Transitivity

English Description

For any three elements, if x is less than or equal to y and y is less than or equal to z, then x is also less than or equal to z."

Mathematical Logic Notation

$\forall x, y, z \quad (x \leq y \wedge y \leq z) \rightarrow x \leq z$

end comment")

(forall (x y z) (if (and (lte x y) (lte y z))
(lte x z)))

(cl:comment "Anti-Symmetry

English Description

For any two elements, if x is less than or equal to y and y is less than or equal to x, then x and y must be the same element.

Mathematical Logic Notation

$\forall (x, y) \quad (x \leq y \wedge y \leq x) \rightarrow x = y$

end comment")

(forall (x y) (if (and (lte x y) (lte y x))
(= x y)))

(cl:comment "Reflexivity

English Description

For any x, x is less than or equal to itself.

Mathematical Logic Notation

$\forall x \quad x \leq x$

end comment")

```
(forall (x) (lte x x))
)
```

A-1.2 Comparability Graphs

A different way of capturing a poset arises in graph theory, with comparability graphs. While the following set of axioms is not equivalent to those for a poset, there exists the following correspondence between these two entities:

- For every poset there exists a unique comparability graph.
- For every comparability graph there exists at least one poset.

It should be noted that the axioms are written in terms of the generic "edge" relation. We also note that we have made use of cl:import statement, which imports axioms that connect the edge relation to less-than-or-equal-to. The axioms for edge are presented immediately following the comparability graph module.

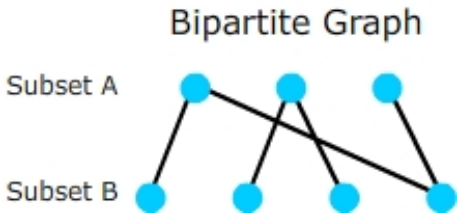
```
(cl:module (comparability_graph)
(cl:import (poset))

(cl:comment "Comparability Graphs
For any directed graph, if transitivity holds, then we have a partially ordered set.
end comment")
        (forall (?x ?y ?z)
                (if     (and    (e ?x ?y) (e ?y ?z))
                        (e ?x ?z)
                )
        )

        (cl:comment "Linking the edge relation to poset lte
```

Whenever an edge exists between two elements, x and y, then either x < y or y <x. ")

```
        (forall (?x ?y)  (iff      (e ?x ?y)
                                   (or (lte ?x ?y) (lte ?y ?x))
                         )
        )
)
```

A-1.3 Bipartite Graphs

A bipartite graph is a special kind of comparability graph that appears in many applications. A common usage is to link two data types to one another, for example, "Social Security Numbers" and "Name." Attributes are commonly implemented as bipartite graphs. It uniquely provides a mapping between two subsets of the elements it ranges over. If we divide the elements into two disjoint subsets, any element in subset A can only be connected to elements in subset B.



(cl:module (Bipartite_Graph)
(cl:import (comparability_graph))
(cl:comment ("Bipartite Graph

English Description

Let A be the set of elements which the relation ranges over. Let B and C be disjoint subsets of A. The relation R, holds only for (R x y), where x ϵ B and y ϵ C or vice versa.

It should be noted that the following axioms require that the ontology designer specify what the two subsets are. Two unary relations are introduced below, “p” and “q” which correspond to disjoint subsets of elements – i.e. “colour” and “material”

Mathematical Logic Notation

$\forall x \quad (x \in P \leftrightarrow x \notin Q)$

$\forall x, y \quad (edge(x, y) \rightarrow (x \in P \wedge y \in Q) \vee (x \in Q \wedge y \in P)$

end comment”)

```
        (forall (?x)      (iff      (p ?x)
                                    (not (q ?x))
                          )
        (forall (?x ?y)   (if       (e ?x ?y)
                                    (or       (and (p ?x) (q ?y))
                                              (and (p ?y) (q ?x))
                                    )
        ))
)
```

A-1.4 Interval Graphs

Interval graphs represent another form of partially ordered sets in graph theory. They have many applications as well, often appearing in the operations research field to model resource allocation.

(cl:module (interval_graph)
(cl:import (comparability_graph))
(cl:comment “Interval Graph

Characterized via C4 free, also the complement is a comparability graph.

English Description

An interval graph is one where each vertex corresponds to an interval, and the edges imply that an overlap exists between the two connected intervals.

Mathematical Logic Notation

let $R = \{I_1, I_2, ..., I_n\}$ be a set of intervals

The interval graph is G = (V,E) such that

$V = \{I_1, I_2, ..., I_n\}$

$\{I_\alpha, I_\beta\} \in E \leftrightarrow I_\alpha \cap I_\beta \neq \emptyset$

end comment")

```
        (forall (?w ?x ?y ?z)
                (if     (and    (e ?x ?y) (e ?y ?w) (e ?w ?z) (e ?z ?x))
                        (or     (e ?w ?x) (e ?y ?z))
                )
                (if     (and    (not (e ?x ?y)) (not (e ?y ?z)))
                        (not (e ?x ?z))
                )
        )
)
```

A-1.5 Cographs

Cographs, which are equivalent to Series-Parallel Graphs constitute class of graphs that correspond to simple rules for their generation. As their name implies they are composed of series-graphs and parallel-graphs. Electric circuits may often be modeled as series-parallel graphs.

```
(cl:module (cograph)
        (cl:import (comparability_graph))
```

(cl:comment “Cograph or Series-Parallel Graphs

English Description

As the name suggests, one way of thinking about these graphs are that they are composed of combinations of series and parallel graphs. Alternatively, they are defined as those graphs in which every connected induced subgraph as a disconnected complement. Yet another definition which is first order expressible states that for any four elements, w,x,y,z the following two statements are equivalent:

w is connected to x and x is connected to y and y is connected to z
w is connected to y or w is connected to z or x is connected z

Mathematical Logic Notation

$\forall(w,x,y,z) \quad (e(w,x) \wedge e(x,y) \wedge e(y,z)) \leftrightarrow (e(w,y) \vee e(w,z) \vee e(x,z))$

end comment”)

```
        (forall (?w ?x ?y ?z)
                (iff    (and    (e ?w ?x) (e ?x ?y) (e ?y ?z)   )
                        (or     (e ?w ?y) (e ?w ?z) (e ?x ?z)   )
                )
        )
)
```

A-1.6 Permutation Graphs

Permutation graphs are those where given a family of line segments connecting two parallel lines in the Euclidean plane; it forms a corresponding intersection graph. Such graphs are often used to explore and/or simulate randomized tests to determine possible associations in large, high-dimensional, sparse data.

(cl:module (permutation graph)

(cl:import (comparability graph))

(cl:comment "Permutation Graph

English Description

Permutation graphs are the intersection graphs for a family of line segments connecting two parallel lines in Euclidean Space. Mathematically, given three elements, x, y and z, we can say:

If x is not connected to y and y is not connected to z, then x cannot be connected to z.

Mathematical Logic Notation

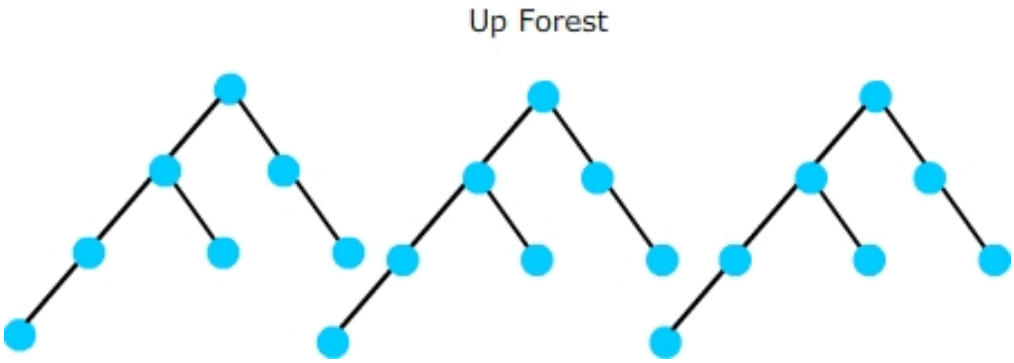
$$\forall(x,y,z) \quad -e(x,y) \wedge -e(y,z) \rightarrow -e(x,z)$$

```
(forall (?x ?y ?z)
        (if     (and    (not (e ?x ?y)) (not (e ?y ?z)))
                (not (e ?x ?z))
        )
)
```

Permutation graphs round out those graphs which are posets that are included in the repository. As noted in chapter 3, there are hundreds of classes of graphs that are extensions of comparability. Many are classified only in terms of a particular forbidden substructure. While these classes of graphs are candidates for inclusion into the repository, they were not deemed significant enough to be included in this thesis.

A-1.7 Up Forest

Trees comprise another prolific branch of partially ordered sets. An "up forest" is a tree with multiple roots, where the roots are located at the "top" of the tree structures. File directory structures in hard drives with multiple partitions are up forests.

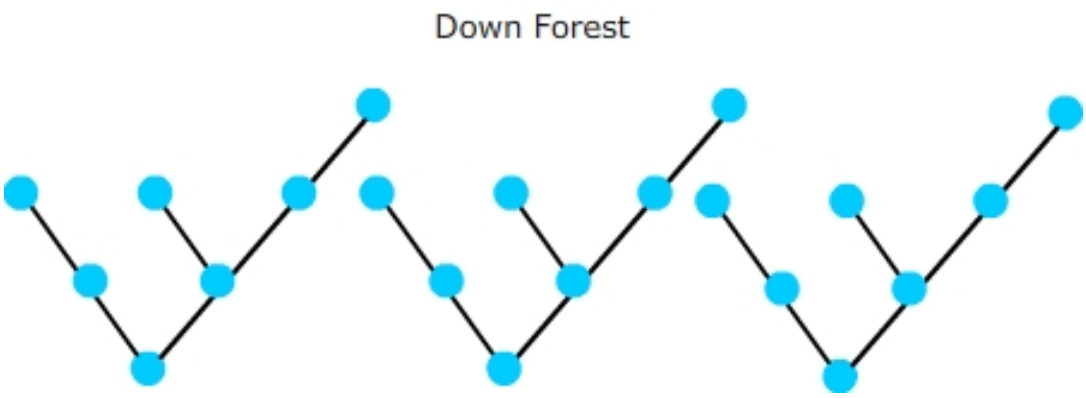



```
(cl:module (up_forest)
(cl:import (poset))
(cl:comment "Up Forests are trees with multiple roots, where the roots are "above" all
their children")
        (forall (?x)
                (iff    (root ?x)
                        (forall (?y)
                                (if     (lte ?y ?x)
                                        (= ?y ?x)
                                )
                        )
        ))
        (forall (?x)
                (exists (?y)
                        (and    (root ?y) (lte ?y ?x))
                )
        )
        (forall (?x)
                (if     (not (root ?x))
                        (exists (?y)
                                (lte ?y ?x)
                                (forall (?z)
```

```
                    (if     (and    (lte ?z ?x)
                                    (lte ?y ?z))
                            (= ?y ?z))
                )
            )
        ))
)
```

A-1.8 Down Forest

These forests are the dual of up forests. Namely they are trees with multiple roots, rooted at the "bottom."

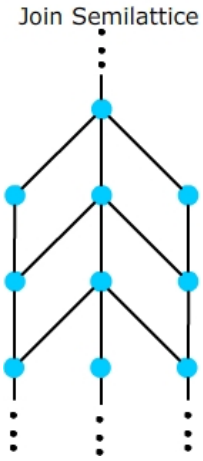



```
(cl:module (down_forest)
(cl:import (poset))
(cl:comment "Down Forests are trees with multiple roots, where the roots are "below" all their children")
    (forall (?x)
        (iff    (root ?x)
                (forall (?y)
                    (if     (lte ?x ?y)
                            (= ?y ?x))
                )
    ))
```

```
        (forall (?x)
                (exists (?y)
                        (and    (root ?y) (lte ?x ?y))
                )
        )
        (forall (?x)
                (if      (not (root ?x))
                         (exists (?y)
                                 (lte ?x ?y)
                                 (forall (?z)
                                         (if      (and    (lte ?x ?z)
                                                          (lte ?z ?y))
                                                  (= ?y ?z))
                                 )
                         )
        ))
)
```

A-1.9 Join Semilattice (JSL)

A join semi lattice asserts that there always exists a join for any two elements; that is to say, any two elements x and y share a common greater element. An example of a join-semilattice might be the organizational hierarchy of a corporation.



(cl:module (joinsemilattice_mj)

(cl:import (lattice_terms))

(cl:import (join))

(cl:comment "Join Semilattice in terms of meet and join

English Description

A join semi lattice is a partial ordering where there always exists the join for any two elements. That is to say, that for any x and y, there is always a third element z, such that z is greater than or equal to both x and y. The join function exhibits the properties of *associativity, idempotency and commutativity.* Consequently, the mub relation must also satisfy these properties.

Note: these properties for join have been defined in the meet module. Here we only assert that there always exists a meet for any two elements.

Mathematical Logic Notation

$\forall(x,y)\exists a \quad a = join(x,y)$

$\forall x \quad x = join(x,x)$

$\forall(x,y,z) \quad join(join(x,y),z) = join(x,(join(y,z))$

end comment")

(forall (?x ?y) (exists (?a) (= ?a (join ?x ?y)))))

(cl:module (joinsemilattice_mm)
(cl:import (lattice_terms))
(cl:import (mub))
(cl:comment "Join Semilattice in terms of minimal upper bound (mub)

English Description

A join semi lattice is a partial ordering where there always exists the join for any two elements. That is to say, that for any x and y, there is always a third element z, such that z is greater than or equal to both x and y. The join function exhibits the properties of *associativity, idempotency and commutativity.* Consequently, the mub relation must also satisfy these properties.

Mathematical Logic Notation

$\forall(x,y)\exists z \quad (mub\ z\ x\ y)$

$\forall(x) \quad (mub\ x\ x\ x)$

$\forall(x,y,z)\exists(a,b,c,d) \quad (mub\ a\ x\ y) \wedge (mub\ b\ a\ z) \wedge (mub\ c\ y\ z) \wedge (mub\ d\ x\ c) \wedge (b = d)$

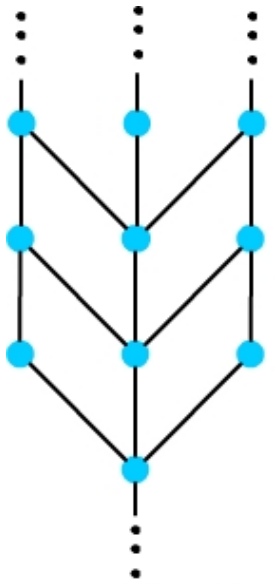
end comment")

```
(cl:comment "there is always a z")
        (forall (x y) (exists (z)          (mub z x y)))

(cl:comment "idempotency")
        (forall (x)       (mub x x x)

(cl:comment "associativity")
        (forall (x y z)
                (exists (a b c d)
                        (and    (mub a x y)
                                (mub b a z)
```

```
                    (mub c y z)
                    (mub d x c)
                    (= b d)
            ))
    )
(cl:comment "commutativity")
    (forall (x y)
            (exists (a b)
                    (and
                            (mub a x y)
                            (mub b y x)
                            (= a b)
    )))

)
```

A-1.10 Meet Semilattice (MSL)

The dual of join semilattices, meet semilattices assert that there always exists a meet for any two elements.



Meet Semilattice

(cl:module (meetsemilattice_mj)

(cl:import (lattice_terms))

(cl:import (meet))

(cl:comment "Meet Semilattice (MSL) in terms of meet and join

English Description

A meet semi lattice is a partial ordering where there always exists the meet for any two elements. That is to say, that for any x and y, there is always a third element z, such that z is less than or equal to both x and y. The meet function exhibits the properties of *associativity, idempotency and commutativity.* Consequently, the mlb relation must also satisfy these properties.

Note: these properties for meet have been defined in the meet module. Here we only assert that there always exists a meet for any two elements.

Mathematical Logic Notation

$\forall(x,y)\exists a \quad a = meet(x,y)$

$\forall x \quad x = meet(x,x)$

$\forall(x,y,z) \quad meet(meet(x,y),z) = meet(x,(meet(y,z))$

end comment")

(forall (?x ?y) (exists (?a) (= ?a (meet ?x ?y))))

)

(cl:module (meetsemilattice_mj)
(cl:import (lattice_terms))
(cl:import (mlb))
(cl:comment "Meet Semilattice (MSL) in terms of maximal lower bound (mlb)

English Description

A meet semi lattice is a partial ordering where there always exists the meet for any two elements. That is to say, that for any x and y, there is always a third element z, such that z is less than or equal to both x and y. The meet function exhibits the properties of *associativity, idempotency and commutativity.* Consequently, the mlb relation must also satisfy these properties.

Mathematical Logic Notation

$\forall(x,y)\exists z \quad (mlb\ z\ x\ y)$

$\forall(x) \quad (mlb\ x\ x\ x)$

$\forall(x,y,z)\exists(a,b,c,d) \quad (mlb\ a\ x\ y) \wedge (mlb\ b\ a\ z) \wedge (mlb\ c\ y\ z) \wedge (mlb\ d\ x\ c) \wedge (b = d)$

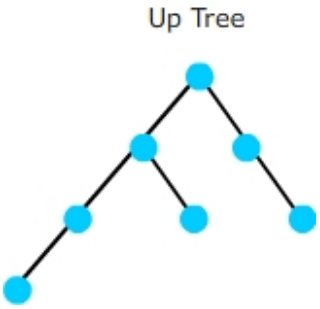
end comment")

```
(cl:comment "there always exists a z")
        (forall (x y) (exists (z)          (mlb z x y)))

(cl:comment "idempotency")
        (forall (x)       (mlb x x x)

(cl:comment "associativity")
        (forall (x y z)
                (exists (a b c d)
                        (and    (mlb a x y)
                                (mlb b a z)
                                (mlb c y z)
                                (mlb d x c)
                                (= b d)
                ))
        )

(cl:comment "commutativity")
        (forall (x y)
                (exists (a b)
                        (and
                                (mlb a x y)
                                (mlb b y x)
                                (= a b)
        )))
)
```

A-1.11 Up Tree

A commonly used data structure corresponds to a tree. An "up tree" is a branching structure with a unique root. The branching occurs such that no descendent has more than

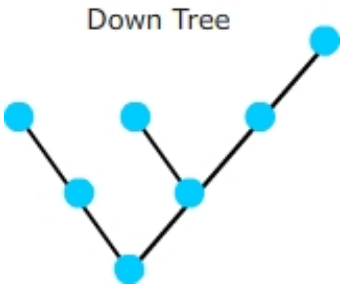
one parent. File directories are the most common examples of these. Another might be tracing “mother” or “father” lineages.




```
(cl:module (up_tree)
        (cl:import (up_forest))
        (forall (?x ?y)
                (if     (and    (root ?x)       (root ?y))
                        (= ?x ?y)
        ))
)
```

A-1.12 Down Tree

Converse of an “up tree,” except the root is now located “below” all other elements.

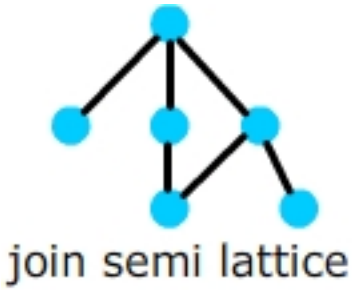



```
(cl:module (down_tree)
(cl:import (down_forest))
```

```
        (forall (?x ?y)
                (if     (and    (root ?x)       (root ?y))
                        (= ?x ?y)
        ))
)
```

A-1.13 Bounded Join Semilattice (BSJL)

This theory introduces the notion of a bound for a semilattice, meaning that there is always a unique element that is less than or equal to every other element. Equivalently, the join of an element called “top” and any other element yields “top”.



(cl:module (boundedjoinsemilattice_mj)
(cl:import (joinsemilattice_mj))
(cl:import (ONE_Top_mj))
(cl:comment “Bounded Join Semilattice in terms of join

English Description

These posets are distinguished by the fact that there always exists a unique *top*. That is to say, there exists a unique element z which is less than or equal to all other elements for which the relation holds. Equivalently the join of *top* with any other element is *top*.

Note this is covered by importing the ONE_Top_mj axioms.

Mathematical Logic

$\forall(x,y)\exists \mathrm{l} \quad (1 = join(1,x) \wedge 1 = join(1,y))$

end comment")

)
(cl:module (boundedjoinsemilattice_mm)
(cl:import (joinsemilattice_mm))
(cl:import (ONE_Top_mm))
(cl:comment "Bounded Join Semilattice in terms of lte

English Description

These posets are distinguished by the fact that there always exists a unique *top*. That is to say, there exists a unique element z which is less than or equal to all other elements for which the relation holds. Equivalently the join of *top* with any other element is *top*.

Mathematical Logic

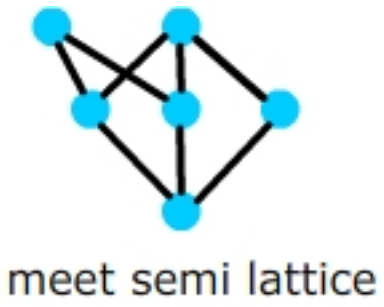
$\forall(x,y)\exists z(x \le z \wedge y \le z) \leftrightarrow \forall w(z \le w) \rightarrow z = w$

end comment")

```
        (forall (x y)
                (and    (lte x ONE)     (lte y ONE))
                        (forall (w)
                                (if     (lte ONE w)
                                        (= ONE w)
                        ))
        ))
)
```

A-1.14 Bounded Meet Semilattice (BMSL)

These are the duals of those above. Here, there is always one element that is less than or equal to all other elements.



(cl:module (boundedmeetsemilattice_mj)
(cl:import (meetsemilattice_mj))
(cl:import (ZERO_Bottom_mj))
(cl:comment "Bounded Meet Semilattice (BMSL)

English Description

These are semilattices such that there always exists a unique "bottom." That is to say that there exists a unique element which is less than or equal to all other elements for which the relation holds.

This module simply imports meet semilattices and the ZERO_Bottom modules to enforce this.

Mathematical Logic Notation

$\forall(x,y)\exists 0 \quad (0 = meet(0,x) \wedge 0 = meet(0,y))$

end comment")
)

(cl:module (boundedmeetsemilattice_mm)
(cl:import (meetsemilattice_mm))
(cl:comment "Bounded Meet Semilattice (BMSL) in terms of mlb and mub

English Description

These are semilattices such that there always exists a unique "bottom." That is to say that there exists a unique element which is less than or equal to all other elements for which the relation holds.

Mathematical Logic Notation

$\forall(x,y)\exists z \quad (z \le x) \wedge (z \le y) \leftrightarrow \forall w(w \le z \rightarrow z = w)$

end comment")

```
        (forall (x y)
                (and    (lte ZERO x)   (lte ZERO y)
                        (forall (w)       (if       (lte w ZERO)  (= w ZERO)))
        ))
)
```

A-1.15 Atomic Cover Property

This branch of the poset hierarchy corresponds to axioms as developed by lattice theorists. Here there is a divergence of notation, where some prefer to introduce the binary functions "meet" and "join." An alternative representation exists using a ternary relation in the guise of "maximal lower bound" (mlb) and "minimal upper bound" (mub). Since they are to be included in the repository, both formulations are presented here.

(cl:module (atomic_covering_property_mj)
(cl:import (joinsemilattice_mj))
(cl:import (meetsemilattice_mj))
(cl:comment "This is the atomic covering property in meet and join")

```
        (forall (x b)    (if      (and    (atom x)
                                          (= (ZERO) (meet b x))
                                 (locover b (join b x))
        ))
)

(cl:module (atomic_covering_property_mm)
(cl:import (joinsemilattice_mm))
(cl:import (meetsemilattice_mm))
(cl:comment "This is the atomic covering property in mlb mub")
        (forall (x b)    (if      (and    (atom x)
                                          (mlb ZERO b x))
                                 (exists (c)
                                          (and    (mub c b x)
                                                  (locover b c))
                                 )
        ))
)
```

A-1.16 Birkhoff's Condition

This class of poset defines weakly modular lattices.

```
(cl:module (Birkhoffs_Condition_mj)
(cl:import (joinsemilattice_mj))
(cl:import (meetsemilattice_mj))
(cl:comment "Birkhoff's Condition in terms of
```

Meet – Join

English Description

If for any two elements, x & y, their meet is the lower cover of both x and y, then x and y both provide a lower cover for their join.

Mathematical Logic Notation

$\forall(x,y) \quad meet(x,y) \prec x \wedge meet(x,y) \prec y \rightarrow x \prec join(x,y) \wedge y \prec join(x,y)$

reference: Stern, M. in *SemiModular Lattices* p. 3, 38")

```
        (forall (?x ?y)  (if      (and    (locover (meet ?x ?y) ?x)
                                          (locover (meet ?x ?y )?y) )
                                  (and    (locover ?x (join ?x ?y))
                                          (locover ?y (join ?x ?y)) )
        ) )
)
```

(cl:module (Birkhoff's_Condition_mm)
(cl:import (joinsemilattice_mm))
(cl:import (meetsemilattice_mm))
(cl:comment "Birkhoff's Condition in terms of mub and mlb

MLB-MUB

English Description

If for any two elements, x & y, their meet is the lower cover of both x and y, then x and y both provide a lower cover for their join.
")

```
(forall (x y)   (exists (z)   (if   (and   (mlb z x y)
                                           (locover z x)
                                           (locover z y))
                                    (exists (w)   (and   (mub w x y)
                                                         (locover x w)
                                                         (locover y w)
                                    ))
                ))
)
)
```

A-1.17 Join-Semi Modular Lattice

A join semi modular lattice or upper-semi modular lattice corresponds to those lattices where modularity holds going "upwards."

```
(cl:module (join_semimodular_lattice_mj)
(cl:import (boundedjoinsemilattice_mj))
(cl:comment "Also known as the join semi modular implication or the Upper Semi
Modular (SM) implication according to Stern, M. in "SemiModular Lattices" p. 37
end comment")

      (forall (x ?y)   (if   (locover (meet x y) x)
                             (locover y (join x y))
      ))
)
```

```
(cl:module (join_semimodular_lattice_mm)
(cl:import (boundedjoinsemilattice_mm))
(cl:comment "Also known as the join semi modular implication or the Upper Semi
Modular (SM) implication according to Stern, M. in "SemiModular Lattices" p. 37
```

```
end comment")

        (forall (x y)
                (exists (a b)
                        (if     (and    (mlb a x y)
                                        (locover a x))
                                (and    (mub b x y)
                                        (locover y b))
                )
        )
)
```

A-1.18 Meet-Semi Modular Lattice

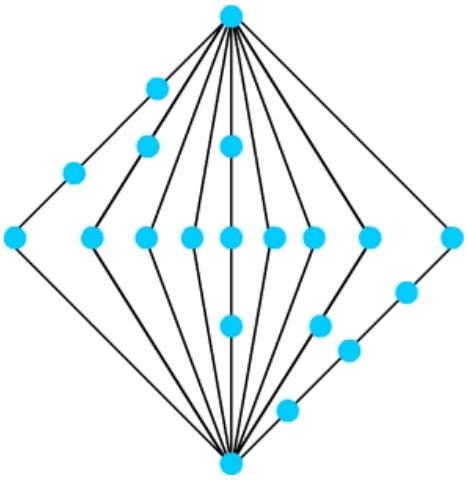
The dual of join semi modular lattices, these lattices define lower semi modularity.

```
(cl:module (meet_semimodular_lattice_mj)
(cl:import (boundedmeetsemilattice_mj))
(cl:comment "Also known as the meet semi modular implication or the Lower Semi
Modular (SM*) implication according to Stern, M. in "SemiModular Lattices" p. 37")

        (forall (x y)   (if     (upcover (join x y) x)
                                (upcover y (meet x y))
        ))
)

(cl:module (join_semimodular_lattice_mm)
(cl:import (boundedmeetsemilattice_mm))
 (cl:comment "Also known as the meet semi modular implication or the Lower Semi
Modular (SM*) implication according to Stern, M. in "SemiModular Lattices" p. 37
end comment")
```

```
        (forall (x y)
                (exists (a b)
                        (if     (and    (mub a x y)
                                        (upcover a x))
                                (and    (mlb b x y)
                                        (upcover y b))
                ))
        )
)
```

A-1.19 Lattice

Combining the notion of the existence of both a top and bottom, yields those partial orders which are lattices. These restrictions also entail novel properties for the meet and join functions.



```
(cl:module (lattice_mj)
(cl:import (boundedjoinsemilattice_mj))
(cl:import (boundedmeetsemilattice_mj))
(cl:comment "Lattice in terms of meet join
```

Meet Join

English description

In addition to the laws enumerated for the meet and join functions in semilattices, the absorption law now also holds.

Mathematical Logic Notation

$\forall(x, y) \quad (meet(\, join(x, y), y) = y$

$\forall(x, y) \quad (\, join(meet(x, y), y) = y$

“

(cl:comment “Absorption laws for meet and join”)

(forall (?x ?y) (= (join (meet ?x ?y) ?y) ?y))

(forall (?x ?y) (= (meet (join ?x ?y) ?y) ?y))

“end comment”)

(cl:module (lattice_mm)

(cl:import (boundedjoinsemilattice_mm))

(cl:import (boundedmeetsemilattice_mm))

(cl:comment “Lattice with minimal upper bound and maximal lower bound

MLB – MUB

English description

In addition to the laws enumerated for the mlb and mub relations in semilattices, the absorption law now also holds.

Math Logic Notation

$\forall(x, y) \quad (meet(\, join(x, y), y) = y$

$\forall(x, y) \quad (\, join(meet(x, y), y) = y$

```
end comment")

(cl:comment "Absorption laws for mlb and mub")

        (forall (x y)     (exists (a b )
                                  (and     ((mlb a x y)     (mub b a y)     (= b y)
                          ))
        )

        (forall (x y)     (exists a b)
                                  (and     ((mub a x y)     (mlb b a y)     (= b y)
                          ))
        )
)
```

A-1.20 Orthosymmetric Lattice

This class of lattice introduces a novel relation, that of modular pair.

```
(cl:module (orthosymmetriclattice_mj)
(cl:import (atomic_covering_property_mj))
(cl:import (ZERO_Top_mj))
(cl:comment ("Orthosymmetric Lattice with meet and join
```

As defined in Stern, M. in "SemiModular Lattices" p. 81

Meet-Join

English Description

A modular pair are any two elements x and y, where x is said to be the modular pair of y if there exists some element c, less than y, and the join of c with the meet of x and y is

equivalent to the meet of y with the join of c and x. An orthosymmetric lattice is one where for any two elements, if x is the modular pair of y and the meet of x and y is the bottom element, then y is also the modular pair of x.

Mathematical Logic Notation

end comment")

```
        (forall (?x ?y)
                (if     (and    (modpair ?x ?y)
                                (= ZERO (meet ?x ?y)))
                        (modpair ?y ?x)
        ))
)
```

```
(cl:module (orthosymmetriclattice_mm)
(cl:import (atomic_covering_property_mm))
(cl:import (ZERO_Top_mm))
(cl:comment ("Orthosymmetric Lattice with mlb mub as defined in Stern, M. in
"SemiModular Lattices" p. 81
```

MLB – MUB

English Description

A modular pair are any two elements x and y, where x is said to be the modular pair of y if there exists some element c, less than y, and the join of c with the meet of x and y is equivalent to the meet of y with the join of c and x. An orthosymmetric lattice is one where for any two elements, if x is the modular pair of y and the meet of x and y is the bottom element, then y is also the modular pair of x.

MLN

end comment")

```
        (forall (x y)
                (if     (and    (modpair x y)
                                (mlb ZERO x y))
                        (modpair y x)
        ))
)
```

A-1.21 Join Pseudo-Complemented Semilattice

Meet-Join

(cl:module (joinPC_semilattice_mj)
(cl:import (boundedjoinsemilattice_mj))
(cl:comment "Join Pseudo-Complemented Semilattice in meet join.

Source = From Stern, M pages 24 and 25.

English Description

It asserts that there exists 1. Moreover, an element t in L has a "meet pseudo-complement – g(t)" if join (t, g) = 1. and (join x t) implies g(t) is lte x . g(t) is the smallest element in the set x, where x is a meet-complement of t.

forall x, exists y, y is p-comp of x if y is comp of x, and forall z where z is a comp of x, y > z.

Mathematical Logic Notation

end comment")

```
        (forall (x)
                (exists (px)
                        (iff    (= px  (mps_complement x))
```

```
                    (forall (z)
                        (if     (and
                                    (=      (ONE) (join px x))
                                    (=      (ONE) (join z x))
                                (lte px z)
                    ))
            ))
        )
)
```

MLB – MUB

(cl:module (joinPC_semilattice_mm)
(cl:import (boundedjoinsemilattice_mm))

Source = From Stern, M pages 24 and 25.

English Description

It asserts that there exists 1. Moreover, an element t in L has a "meet pseudo-complement – g(t)" if join (t, g) = 1. and (join x t) implies g(t) is lte x . g(t) is the smallest element in the set x, where x is a meet-complement of t.

forall x, exists y, y is p-comp of x if y is comp of x, and forall z where z is a comp of x, y > z.

Mathematical Logic Notation

```
end comment")
        (forall (x)
            (exists (px)
                (iff    (= px (mps_complement x))
                        (forall (z)
```

```
                                        (if     (and
                                                        (mub (ZERO) px x)
                                                        (mub (ZERO) z x))
                                                (lte px z)
                                ))
                        ))
                )
)
```

A-1.22 Meet Pseudo-Complemented Semilattice

Meet-Join

(cl:module (meetPC_semilattice_mj)
(cl:import (boundedmeetsemilattice_mj))
(cl:comment "Pseudo-Complemented Meet Semilattice in meet-join.

From Stern, M pages 24 and 25.

English Description

It asserts that there exists 0. Moreover, an element t in L has a "meet pseudo-complement – g(t)" if meet (t, g) = 0. and (meet x t) implies x is lte g(t) . g(t) is the largest element in the set x, where x is a meet-complement of t.

forall x, exists y, y is p-comp of x if y is comp of x, and forall z where z is a comp of x, y > z.

Mathematical Logic Notation

end comment")

```
        (forall (x)
                (exists (px)
                        (iff      (= px  (mps_complement x))
```

```
                    (forall (z)
                        (if     (and
                                    (=      (ZERO)      (meet px x))
                                    (=      (ZERO)      (meet z x))
                                (lte z px)
                    ))
            ))
        )
)
```

MLB-MUB

(cl:module (meetPC_semilattice_mm)

(cl:import (boundedmeetsemilattice_mm))

(cl:comment "Pseudo-Complemented Meet Semilattice in meet-join.

From Stern, M pages 24 and 25.

English Description

It asserts that there exists 0. Moreover, an element t in L has a "meet pseudo-complement – g(t)" if meet (t, g) = 0. and (meet x t) implies x is lte g(t) . g(t) is the largest element in the set x, where x is a meet-complement of t.

forall x, exists y, y is p-comp of x if y is comp of x, and forall z where z is a comp of x, y > z.

Mathematical Logic Notation

end comment")

```
        (forall (x)
            (exists (px)
                (iff    (= px (mps_complement x))
                        (forall (z)
```

```
                                    (if     (and
                                                (mlb (ZERO) px x)
                                                (mlb (ZERO) z x))
                                            (lte z px)
                                ))
                        ))
                )
)
```

A-1.23 Double p-Lattice

This class of partial orders simply asserts that there exists both a join and meet pseudo-complement for each element.

Meet – Join

```
(cl:module (double_plattice_mj)
(cl:import (joinPC_semilattice_mj))
(cl:import (meetPC_semilattice_mj)
)
```

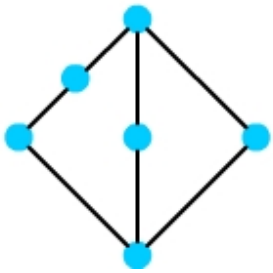
MLB – MUB

```
(cl:module (double_plattice_mm)
(cl:import (joinPC_semilattice_mm))
(cl:import (meetPC_semilattice_mm)
)
```

A-24 Complemented Lattice

These are lattices where for any element, there always exists its “complement” where the only element they share below is the bottom (ZERO) and the only element they share above is top (ONE).

Complemented Lattice



(cl:module (complementedlattice_mj)
(cl:import (lattice_mj))
(cl:import (complement_mj))
(cl:comment “Complemented Lattice

according to Stern, M. in “SemiModular Lattices” p. 9”)

(forall (?x) (exists (?a) (complement ?a ?x)))
)

(cl:module (complementedlattice_mm)
(cl:import (lattice_mm))
(cl:import (complement_mm))
(cl:comment “Complemented Lattice

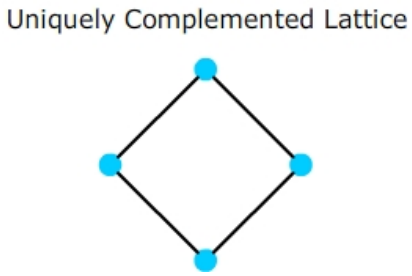
according to Stern, M. in “SemiModular Lattices” p. 9”)

(forall (?x) (exists (?a) (complement ?a ?x)))

)

A-1.25 Uniquely Complemented Lattice

A uniquely complemented lattice asserts that there always exists a unique complement for any element in the set. It is more restrictive than a simple complemented lattice. An example of these lattices might be the factoring decomposition for a number.



Meet Join

(cl:module (uniquely_complemented_lattice_mj)

(cl:import (complementedlattice_mj)

(cl:comment "Uniquely Complemented Lattice in terms of meet and join,

English Description

For any element in the lattice, there exists exactly one other element that is its complement

MLN

end comment")

```
        (forall (x)
                (exists (x')
                        (= x' (fcomplement x))
                )
        )
)
```

MLB – MUB

(cl:module (uniquely_complemented_lattice_mm)

(cl:import (complementedlattice_mm))

(cl:comment "Uniquely Complemented Lattice in terms of mlb-mub,

English Description

For any element in the lattice, there exists exactly one other element that is its complement

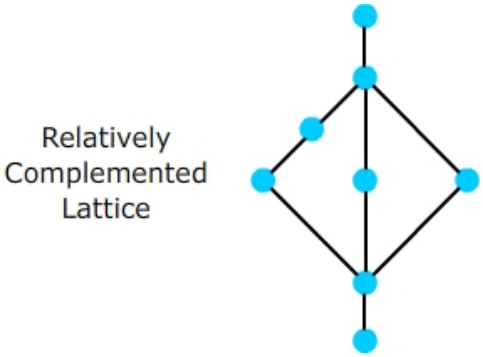
MLN

end comment")

```
        (forall (x)
                (exists (x')
                        (= x' (fcomplement x))
                )
        )
)
```

A-1.26 Relatively Complemented Lattice

These lattices have complements for every interval in the lattice.



Meet-Join

(cl:module (relatively_complemented_lattice_mj)
(cl:import (complementedlattice_mj))
(cl:comment "Relatively Complemented Lattice – RCL

Each interval of a lattice is complemented as a sublattice, according to Stern, M page 10 and 29.

English Description

For a given interval [a,b], an element x in that interval has a complement x', if the common top element is b and the common bottom element is a.

Mathematical Notation

end comment")

```
        (forall (a b x)
                (if     (and    (lte a b) (lte a x) (lte x b))
                        (exists (x')
                                (iff    (rcomplement x' x)
                                        (and    (lte a x')
                                                (lte x' b)
                                                (= a (meet x x'))
                                                (= b (join x x'))
                        )))
        ))
)
```

MLB-MUB

(cl:module (relatively_complemented_lattice_mm)
(cl:import (complementedlattice_mm))

(cl:comment “Relatively Complemented Lattice – RCL

Each interval of a lattice is complemented as a sublattice, according to Stern, M page 10 and 29.

English Description

For a given interval [a,c], an element b in that interval has a complement x, if the common top element is c and the common bottom element is a.

Mathematical Notation

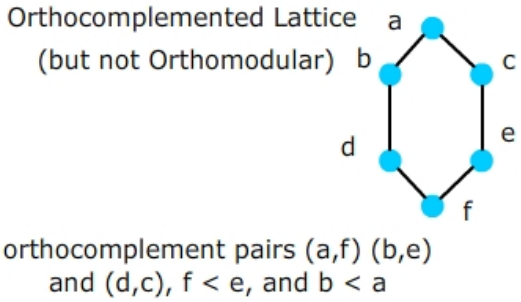
end comment”)

```
(forall (a b x)
        (if     (and    (lte a b) (lte a x) (lte x b))
                (exists (x’)
                        (iff    (rcomplement x’ x)
                                (and    (lte a x’)
                                        (lte x’ b)
                                        (mlb a x x’))
                                        (mub b x x’))
                                        (forall (c)
                                                (if (mlb c x x’) (= a c)))
                                        (forall (c)
                                                (if mub c x x’) (= a b)))
                ))
))
```

A-1.27 Ortholattice (Orthocomplemented Lattice)

These lattices have the property whereby every element has a corresponding orthocomplement. That is to say, for any x there is its orthocomplement y, where the

meet of x and y is ZERO, their join is ONE and the orthocomplement of y is x. Moreover, for any x and y is x less than y, then the orthocomplement of y is less than the orthocomplement of x.



Meet – Join

(cl:module (ortholattice_mj)

(cl:import (lattice_mj))

(cl:import (orthocomplement_mj))

(cl:comment "These are axioms for an ortholattice (also known as Orthocomplemented Lattice or OC lattice using meet and join")

(forall (x) (exists ox) (= ox (o_complement x)))

)

MLB – MUB

(cl:module (ortholattice_mm)

(cl:import (lattice_mm))

(cl:import (orthocomplement_mm))

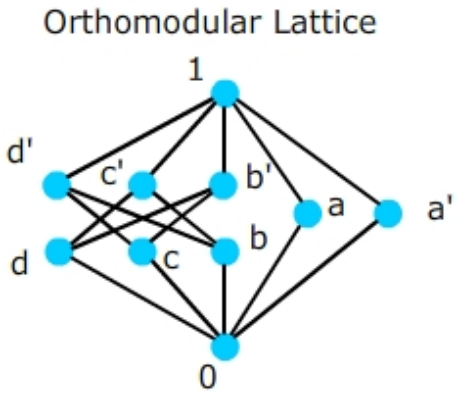
(cl:comment "These are the axioms for an ortholattice (aka Orthocomplemented Lattice or OC Lattice using the mlb and mub relations")

(forall (x) (exists ox) (= ox (o_complement x))))

)

A-1.28 Orthomodular Lattice

An extension of the orthocomplemented lattice, this class of poset asserts that if for any two pairs, if x and y are orthocomplemented, they also form a modular pair.



Meet – Join

(cl:module (orthomodular_lattice_mj)

(cl:import (ortholattice_mj))

(cl:comment "Orthomodular lattice – in meet-join.

This class of lattice asserts that for any two elements, x and y, if x is the orthocomplement of y then x and y are a modular pair.")

```
        (forall (x)
                (exists (ox)
                        (if      (= ox (o_complement x))
                                 (modpair x ox)
                ))
        )
)
```

MLB – MUB

(cl:module (orthomodular_lattice_mm)

(cl:import (ortholattice_mm))

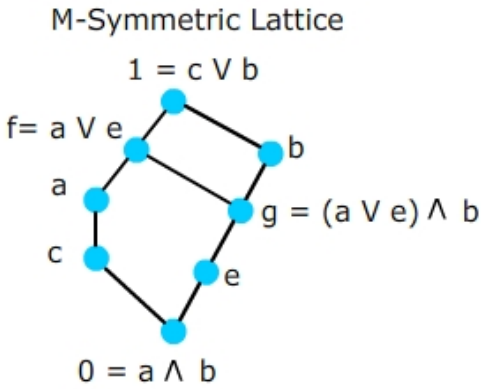
(cl:comment "Orthomodular lattice – in mlb mub.

This class of lattice asserts that for any two elements, x and y, if x is the orthocomplement of y then x and y are a modular pair.")

```
(forall (x)
        (exists (ox)
                (if     (= ox (o_complement x))
                        (modpair x ox)
                ))
        )
)
```

A-1.29 M-Symmetric Lattice

These are lattices where if x is the modular pair of y then y is also the modular pair of x.



Meet – Join

(cl:module (msymmetric_lattice_mj)

(cl:import (join_semimodular_lattice_mj))

(cl:comment "M-Symmetric Lattice

According to Stern, M. in "SemiModular Lattices" p. 66

English Description

For any two elements, if x is the modular pair of y, then y is the modular pair of x.

Mathematical Logic Notation

end comment")

(forall (?x ?y) (iff (modpair ?x ?y) (modpair ?y ?x)))
)

MLB – MUB

(cl:module (msymmetric_lattice_mm)
(cl:import (join_semimodular_lattice_mm))
(cl:comment "M-Symmetric Lattice in terms of MLB-MUB

According to Stern, M. in "SemiModular Lattices" p. 66

English Description

For any two elements, if x is the modular pair of y, then y is the modular pair of x.

Mathematical Logic Notation

end comment")

(forall (?x ?y) (iff (modpair ?x ?y) (modpair ?y ?x)))
)

A-1.30 M-Symmetric Lattice*

This is the dual of those above. That is to say, if x is the dual modular pair of y, then y is the dual modular pair of x.

Meet – Join

(cl:module (m*symmetric_lattice_mj)
(cl:import (meet_semimodular_lattice_mj))
(cl:comment "M*-Symmetric Lattice in meet-join

English Description

These lattices are the dual of the M-Symmetric Lattices. A dual modular pair is modular in the dual lattice.

Mathematical Logic Notation

end comment")
(forall (x y) (iff (dmodpair x y) (dmodpair y x)))
)

MLB – MUB

(cl:module (m*symmetric_lattice_mm)
(cl:import (meet_semimodular_lattice_mm))
(cl:comment "M*-Symmetric Lattice in mlb-mub

English Description

These lattices are the dual of the M-Symmetric Lattices. A dual modular pair is modular in the dual lattice.

Mathematical Logic Notation

end comment")
(forall (x y) (iff (dmodpair x y) (dmodpair y x))))

A-1.31 Cross Symmetric Lattice

Cross symmetric lattices are an extension of the M-Symmetric lattice. All cross-symmetric lattices are M-symmetric, but they are not M*-symmetric.

Meet – Join

```
(cl:module (cross_symmetriclattice_mj)
(cl:import (msymmetric_lattice_mj))
(cl:comment "Cross Symmetric Lattice in meet join. According to Stern, M. in
"SemiModular Lattices" p. 83
end comment")

(forall (?x ?y?)
        (if       (modpair ?x ?y) (dmodpair ?y ?x))
)
)
```

MLB - MUB

```
(cl:module (cross_symmetriclattice_mm)
(cl:import (msymmetric_lattice_mm))
(cl:comment "Cross Symmetric Lattice in mlb – mub. According to Stern, M. in
"SemiModular Lattices" p. 83
end comment")

(forall (?x ?y?)
        (if       (modpair ?x ?y) (dmodpair ?y ?x))
)
)
```

A-1.32 Dual Cross Symmetric Lattice

Meet – Join

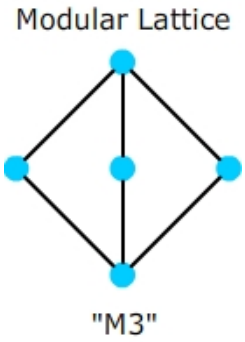
```
(cl:module (dcross_symmetriclattice_mm)
(cl:import (msymmetric_lattice_mj))
(cl:comment "Dual Cross Symmetric Lattice in meet join. According to Stern, M. in
"SemiModular Lattices" p. 83
end comment")

        (forall (?x ?y?)
                (if       (dmodpair ?x ?y) (modpair ?y ?x))
        )
)
```

MLB – MUB

```
(cl:module (dcross_symmetriclattice_mm)
(cl:import (msymmetric_lattice_mm))
(cl:comment "Dual Cross Symmetric Lattice in mlb mub. According to Stern, M. in
"SemiModular Lattices" p. 83
end comment")

        (forall (?x ?y?)
                (if       (dmodpair ?x ?y) (modpair ?y ?x))
        )
)
```

A-1.33 Modular Lattice

A modular lattice introduces even more properties for the meet/join or mlb/mub functions and relations respectively.



(cl:module (modular_lattice_mj)
(cl:import (meet_semimodular_lattice_mj))
(cl:import (join_semimodular_lattice_mj))

(cl:comment "Modular Lattice in Meet Join

Meet Join

English Description

A non-conservative extension of both the upper and lower semimodular lattices, according to Stern, M. in "SemiModular Lattices" p. 10)

Mathematical Logic Notation

"end comment")

```
        (forall (x y)
                (exists (z)
                        (iff    (lte z y)
                                (= (join z (meet x y)) (meet (join z x) y))
                ))
        )
)
```

(cl:module (modular_lattice_mm)
(cl:import (meet_semimodular_lattice_mm))
(cl:import (join_semimodular_lattice_mm))

(cl:comment “Modular Lattice in mlb-mub

MLB – MUB

English Description

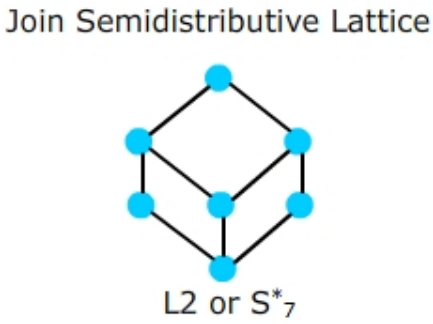
A non-conservative extension of both the upper and lower semimodular lattices, according to Stern, M. in “SemiModular Lattices” p. 10)

Mathematical Logic notation

end comment”)

```
        (forall (x y) (exists (z)
                        (iff     (lte z y)
                                 (exists (a b c d)
                                         (and    (mlb a x y)     (mub b z a)
                                                 (mub c z x)     (mlb d c y)
                                                 (= b d))
                                 )
                        ))
        )
)
```

A-1.34 Join Semi Distributive Lattice (JSDL)



Meet Join

(cl:module (join_semidist_lattice_mj)

(cl:import (boundedjoinsemilattice_mj))

(cl:comment “Join Semi Distributive Lattice (JSDL) in terms of meet and join according to Stern, M. in “SemiModular Lattices” p. 18

English Description

Mathematical Logic Notation

end comment”)

```
        (forall (?x ?y ?z )
                (if     (= (join ?x ?y) (join ?x ?z))
                        (= (join ?x ?y) (join ?x (meet ?y ?z)))
                )
        )
)
```

MLB-MUB

(cl:module (join_semidist_lattice_mm)

(cl:import (boundedjoinsemilattice_mm))

(cl:comment “Join Semi Distributive Lattice (JSDL) in terms of mlb and mub according to Stern, M. in “SemiModular Lattices” p. 18

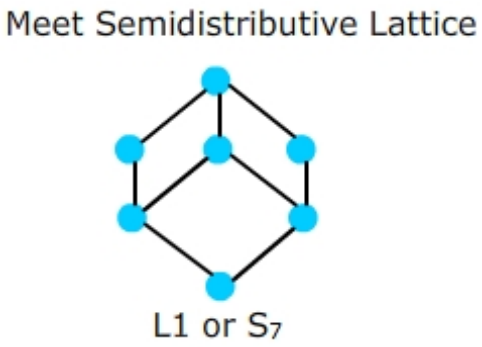
English Description

Mathematical Logic Notation

end comment”)

```
        (forall (x y z)
                (exists (a b c d)
                        (if     (and    (mub a x y)     (mub b x z)     (= a b))
                                (and    (mlb c y z)     (mub d x c)     (= a d))
                ))))
```

A-1.35 Meet Semi Distributive Lattice (MSDL)



Meet Join

(cl:module (meet_semidist_lattice_mj)

(cl:import (boundedmeetsemilattice_mj))

(cl:comment "Meet Semi Distributive Lattice (MSDL) in terms of Meet and Join, according to Stern, M. in "SemiModular Lattices" p. 18

English Description

Mathematical Logic Notation

end comment")

```
        (forall (?x ?y ?z)
                (if     (= (meet ?x ?y) (meet ?x ?z))
                        (= (meet ?x ?y) (meet ?x (join ?y ?z)))
                        )
                )
        )
)
```

MLB-MUB

(cl:module (meet_semidist_lattice_mm)

(cl:import (boundedmeetsemilattice_mm))

(cl:comment “Meet Semi Distributive Lattice (MSDL) in terms of mlb and mub, according to Stern, M. in “SemiModular Lattices” p. 18

English Description

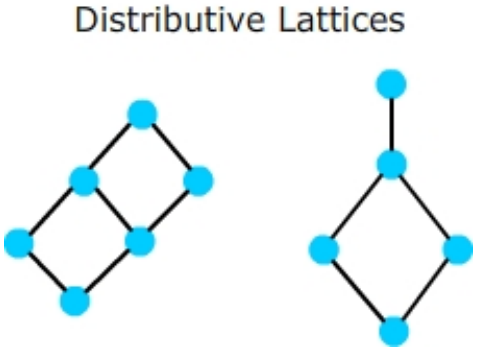
Mathematical Logic Notation

end comment”)

```
        (forall (x y z)
                (exists (a b c d)
                        (if     (and    (mlb a x y) (mlb b x z) (= a b))
                                (and    (mub c y z) (mlb d x c) (= a d))
                ))
        )
)
```

A-1.36 Distributive Lattice

This class of theories asserts that when a set is partially ordered and either the functions of meet-join or the relations mlb-mub are defined, they satisfy the Distributivity Laws. Examples of these lattices are abundant, as they can be used to represent structures for sets that make use of Union and Intersection.



Meet Join

(cl:module (distributive_lattice_mj)

(cl:import (lattice_mj))

(cl:comment "Distributive Lattice in meet join

English Description

These lattices assert that the law of distributivity holds for the meet-join functions.

According to Stern, M. in "SemiModular Lattices" p. 8

Mathematical Logic Notation

end comment")

```
        (forall (x y z)
                (=      (meet x (join y z))
                        (join (meet x y) (meet x z)))
        )

        (forall (x y z)
                (=      (join x  (meet y z)
                        (meet (join x y) (join x z)))
        )
)
```

MLB – MUB

(cl:module (distributive_lattice_mm)

(cl:import (lattice_mm))

(cl:comment "Distributive Lattice in mlb mub

English Description

These lattices assert that the law of distributivity holds for the mlb-mub relations.

According to Stern, M. in "SemiModular Lattices" p. 8

Mathematical Logic Notation

```
end comment")
        (forall (x y z)
                (exists (a b c d e)
                (and    (mub a y z) (mlb b x a) (mlb c x y) (mlb d x z) (mub e c d) (= b e))
                )
        )

        (forall (x y z)
                (exists (a b c d e)
                (and    (mlb a y z) (mub b x a) (mub c x y) (mub d x z) (mlb e c d) (= b e))
                )
        )
)

(cl:module (distributive_lattice_FB)
(cl:comment "Forbidden Substructures

These are the forbidden substructures for distributive lattices; their existence implies that
the given ordering is not distributive."

(or
        (cl:comment "M3 – forbidden sublattice")
        (not    (exists (top bottom a b c)
                        (and    (lte a top) (lte b top) (lte c top)
                                (lte bottom a) (lte bottom b) (lte bottom c))
        ))
        (cl:comment "N5 – forbidden sublattice")
         (not   (exists (top bottom a b c)
                        (and    (lte a top) (lte b top)
                                (lte c b)
```

(lte bottom a) (lte bottom b)

))

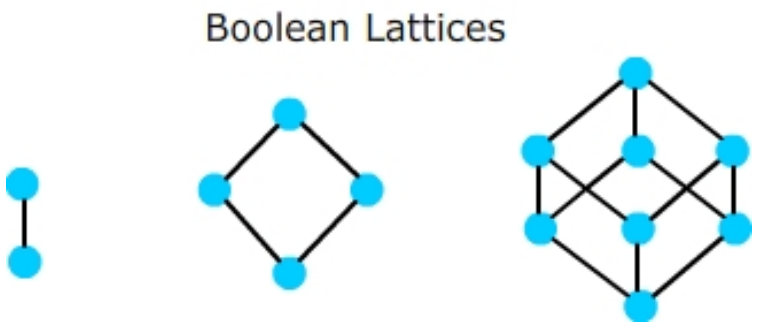
)

)

"end forbidden substructures")

A-1.37 Boolean Lattice

This class of lattice asserts that Associativity, Commutativity, Absorption, Distributivity hold for the meet-join functions or the mlb-mub relations. Moreover, they also assert that a unique complement for any element also exists. Boolean algebras are mappable to Boolean lattices.



Meet - Join

(cl:module (Boolean_lattice_mj)

(cl:import (distributive_lattice_mj))

(cl:import (uniquely_complemented_lattice_mj))

(cl:comment "Boolean Lattices

Assert the following properties for the meet-join functions:

Associativity

Commutativity

Absorption

Distributivity

They also assert that every element in the lattice has at least one complement, where the only common element above any element and its complement is the top or ONE. Conversely the only shared common element below is the bottom or ZERO.

The conditions on the meet-join are satisfied by importing distributive lattice. Importing unique complements ensures that the resultant lattice will be Boolean.")

)

MLB – MUB

(cl:module (Boolean_lattice_mm)
(cl:import (distributive_lattice_mm))
(cl:import (uniquely_complemented_lattice_mm))
(cl:comment "Boolean Lattices

Assert the following properties for the mlb-mub relations:

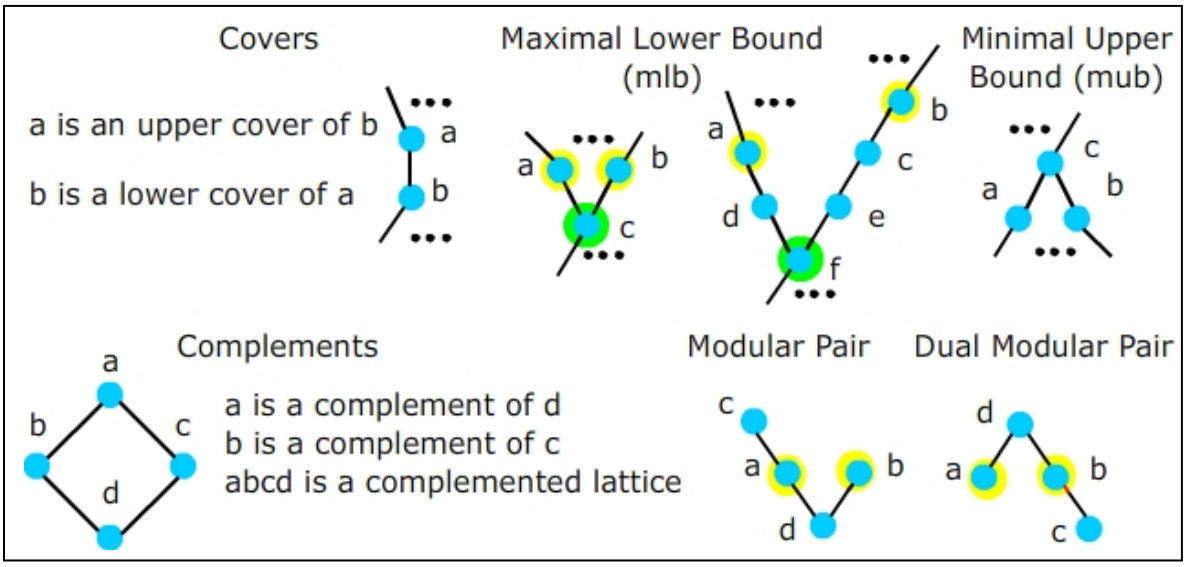
Associativity
Commutativity
Absorption
Distributivity

They also assert that every element in the lattice has at least one complement, where the only common element above any element and its complement is the top or ONE. Conversely the only shared common element below is the bottom or ZERO.

The conditions on the mlb-mub are satisfied by importing distributive lattice. Importing unique complements ensures that the resultant lattice will be Boolean."))

A-2 Basic Lattice Terms

These modules define relations and functions that appear in lattice theory. They are



imported to the relevant modules where lattices with these properties are defined.

(cl:module (lattice_terms)

(cl:comment "This module simply collects some terms that are defined in their own modules")

(cl:import (comparable))

(cl:import (atom))

(cl:import (minimal))

(cl:import (covers))

)

A-2.1 Comparable

The comparable relation asserts that if x and y are comparable then an ordering exists between them. In this regard, it is analogous to the edge relation.

(cl:module (comparable)

```
(cl:import (poset))
(cl:comment "This module defines the comparable relation in terms of less than or equal to")
(forall (x y)    (iff    (comparable x y)
                         (or     (lte x y) (lte y x))
))
```

A-2.2 Atom

Atom is a unary relation which asserts that if x is an atom, then the only element below it is the minimal element.

```
(cl:module (atom)
(cl:import (poset))
(cl:import (minimal))
(cl:import (covers))

        (forall (?x)
                (iff    (atom ?x)
                        (exists (?y)
                                (and    (minimal ?y) (upcover ?x ?y)
                        ))
        ))
)
```

A-2.3 Minimal

Analogous to ZERO, however defined as a unary relation. For an element to be minimal, it asserts that there is no other element "below" it.

```
(cl:module (minimal_element)
(cl:import (poset))
```

```
        (forall (x)     (iff    (minimal x)
                                (exists (y)     (if     (lte y x)
                                                        (= y x)
                                )))
        ))
)
```

A-2.4 Covers

These two modules define two binary relations, “locover” and “upcover.” If x is the lower cover of y, it asserts that there exist no other elements between x and y, while x is less than or equal to y. Conversely, for x to be an upper cover of y, it asserts that x is larger than or equal to y, and there are no other elements in between them.

```
(cl:module (covers)
(cl:import (poset))
(cl:comment “Lower Cover”)
        (forall (x y)
                (iff    (locover x y)
                        (and    (lte x y)
                                (forall (z)
                                        (if     (and    (lte x z)
                                                        (lte z y)
                                                        (not (= z y)))
                                                (= x z)
                                        ))
        )))
(cl:comment “Upper Cover”)
        (forall (x y)
                (iff    (upcover x y)
                        (and    (lte y x)
```

```
                    (forall (z)
                        (if     (and    (lte y z)
                                        (lte z x)
                                        (not (= z y))
                                (= x z)
                    ))
        )))
)
```

A-3 Additional Terms for Lattices

A-3.1 ZERO and ONE

These two modules define the existence of bottommost and topmost elements respectively.

```
(cl:module (ZERO_Bottom_mj)
(cl:comment "Defines a lower bound for a partial order, where the function (ZERO)
selects the unique element that is smaller than all other elements")

(forall (?x) (=  (join ?x ZERO)        ?x))
(forall (?x) (=  (meet ?x ZERO)        ZERO))
)

(cl:module (ZERO_Bottom_mm)
(cl:import (poset))
(cl:comment "Defines a lower bound for a partial order, where the function (ZERO)
selects the unique element that is smaller than all other elements")
 (forall (?x) (lte ZERO ?x))
)
```

```
(cl:module (ONE_Top_mj)
        (cl:comment "Defines an upper bound for a partial order, where function (ONE)
selects the unique element that is larger than all other elements in the set")

(forall (?x) (=  (join ?x ONE)          ONE))
(forall (?x) (=  (meet ?x ONE)          ?x))

(cl:module (ONE_Top_mm)
(cl:import (poset))
        (cl:comment "Defines an upper bound for a partial order, where function (ONE)
selects the unique element that is larger than all other elements in the set")

(forall (?x) (lte ?x ONE))

)
```

A-3.2 Complement

Here the notion of a complement is defined both as a relation and a function. Complement as a function simply asserts that two elements are complements of one another if their joins is equal to ONE and their meet is ZERO. Complement as a function asserts that each element has a unique complement. It is defined in terms of complement as a binary relation.

```
(cl:module (complement_mj)
(cl:import (meet_join)
(cl:import (ZERO_Bottom))
(cl:import (ONE_Top))
(cl:comment "These are the axioms for a relation that determines whether an element is a
complement of another in terms of meet and join")
        (forall (x y)
```

```
            (iff     (complement x y)
                     (and    (= (ZERO) (meet x y))
                             (= (ONE) (join x y))
        )))

        (forall (x)      (iff     (fcomplement x)
                                  (exists  (a)
                                           (and    (= a (fcomplement x))
                                                   (complement a x)
                                                   (forall (b)
                                                           (if      (complement b x)
                                                                    (= a b)
                                                   ))
                                  ))
        ))
)
(cl:module (complement_mm)
(cl:import (mlb-mub))
(cl:import (ZERO_Bottom))
(cl:import (ONE_Top))
(cl:comment "These are the axioms for a relation that determines whether an element is a
complement of another in terms of mlb and mub")
        (forall (x y)
            (iff     (complement x y)
                     (and    (mlb (ZERO) x y)
                             (mub (ONE) x y)
        )))
        (forall (x)      (iff     (fcomplement x)
                                  (exists  (a)
                                           (and    (= a (fcomplement x))
                                                   (complement a x)
```

```
                                        (forall (b)
                                            (if     (complement b x)
                                                    (= a b)
                                        ))
                                ))
        ))
)
```

A-3.3 Orthocomplement

```
(cl:module (orthocomplement_mj)
(cl:import (poset))
(cl:comment "Defining the notion of orthocomplementation on L as defined in M Stern,
Semi Modular Lattices page 11")
        (forall (x)     (exists ox)
                        (iff    (= ox (o_complement x)
                                (and    (= (ZERO) (meet x ox))
                                        (= (ONE) (join x ox))
                                        (forall (b) (if   (lte x b)
                                                          (lte ox (o_complement b)))
                                        (= x (o_complement ox))
                        )))
        )
)

(cl:module (orthocomplement_mm)
(cl:import (poset))
(cl:comment "Defining the notion of orthocomplementation on L as defined in M Stern,
Semi Modular Lattices page 11")
        (forall (x)     (exists ox)
                        (iff    (= ox (o_complement x)
```

```
                                    (and    (mlb (ZERO)  x ox)
                                            (mub (ONE) x ox))
                                            (forall (b) (if   (lte x b)
                                                              (lte ox (o_complement b)))
                                            (= x (o_complement ox))
                        )))
        )
)
```

A-3.4 Orthogonal

Another property that has been noticed and deemed worth naming is that of one element being orthogonal to another. Orthogonality in the context of lattices means that if x and y are orthogonal to one another, there exists a third element z, which is the orthocomplement of y, and x < z.

```
(cl:module (orthogonal_mj)
(cl:import (orthocomplement_mj))
(cl:import (poset))
(cl:comment "This module defines the notion of an element being orthogonal to another
in terms of meet and join")
        (forall (x y)
                (iff    (orthogonal x y)
                        (exists (z)
                                (and    (= z (o_complement y))
                                        (lte x z)
                        ))
        ))
)

(cl:module (orthogonal_mm)
```

```
(cl:import (ortho_complement_mm))
(cl:import (poset))
(cl:comment "This module defines the notion of an element being orthogonal to another
in terms of mlb and mub")
        (forall (x y)
                (iff    (orthogonal x y)
                        (exists (z)
                                (and    (= z (ortho_complement y))
                                        (lte x z)
                        ))
        ))
)
```

A-3.5 Modular Pair

```
(cl:module (mod_pair_mj)
(cl:import (meet-join))
(cl:import (poset))
(cl:comment "Defines the notion of a Modular pair (right modular pair) in terms of meet
and join:")
        (forall (x y)
                (iff    (modpair x y)
                        (forall (z)
                                (and    (lte z x)
                                        (=      (join z (meet x y))
                                                (meet (join z x) y))
                        )))
        ))
)

(cl:module (mod_pair_mm)
```

```
(cl:import (mlb-mub))
(cl:import (poset))
(cl:comment "Defines the notion of a Modular pair (right modular pair) in terms of mlb
and mub")
        (forall (x y)
                (iff    (modpair x y)
                        (forall (z)
                                (exists (a b c d)
                                        (and    (lte z x)
                                        (mlb a x y) (mub b z a) (mub c z x) (mlb d c y)
                                        (= b d)
                                ))
                        )
        ))
)
```

A-3.6 Dual Modular Pair

```
(cl:module (dmod_pair_mj)
(cl:import (meet-join))
(cl:import (poset))
(cl:comment "Defines the notion of a Modular* pair (left modular pair) in terms of meet
and join:")
        (forall (x y)
                (iff    (dmodpair x y)
                        (forall (z)
                                (and    (lte x z)
                                        (=      (meet z (join x y))
                                                (join (meet z x) y))
                        )))
        ))
```

```
)

(cl:module (dmod_pair_mm)
(cl:import (mlb-mub))
(cl:import (poset))
(cl:comment "Defines the notion of a Modular* pair (left modular pair) in terms of mlb
and mub")
        (forall (x y)
                (iff    (dmodpair x y)
                        (forall (z)
                                (exists (a b c d)
                                        (and    (lte z x)
                                        (mub a x y) (mlb b z a) (mlb c z x) (mub d c y)
                                        (= b d)
                                ))
                        )
        ))
)
```

A-3.7 Meet and Join

```
(cl:module (meet)
(cl:comment "Idempotency")
(forall (x)       (= x (meet x x))

(cl:comment "Associativity")

(forall (x y z)  (=       (meet x (meet y z))     (meet z (meet x y)) ) )

(cl:comment "Commutativity")
```

```
(forall (x y)     (=        (meet x y)       (meet y x)))
)

(cl:module (join)
(cl:comment "Idempotency")
(forall (x)       (= x (join x x)

(cl:comment "Associativity")
(forall (x y z)  (=        (join (join y z) x)       (join (join x y) z)))

(cl:comment "Commutativity")
(forall (x y)     (=        (join x y)       (join y x)))

)
```

A-3.8 Maximal Lower Bound and Minimal Upper Bound

```
(cl:module (mlb)
(cl:import (poset))
(cl:comment "Defining maximal lower bound mlb")
        (forall (?x ?y ?z)        (iff      (mlb ?z ?x ?y))
                                            (and     (lte ?z ?x)
                                                     (lte ?z ?y)
                                                     (forall (?i) (if  (and (lte ?i ?x) (lte ?i ?y))
                                                                       (lte ?i ?z)
                                                     ))
        )))
)

(cl:module (mub)
(cl:import (poset))
(cl:comment "Defining minimal upper bound mub")
```

```
        (forall (?x ?y ?z)      (iff    (mub ?z ?x ?y))
                                        (and    (lte ?x ?z)
                                                (lte ?y ?z)
                                                (forall (?i) (if  (and (lte ?x ?i) (lte ?y ?i))
                                                                  (lte ?z ?i)
                                                ))
        )))
)
```

A-4 Basic Properties

This collection of modules consists of properties that apply to binary relations. They are very generic properties that may be used in many other applications and extensions beyond that of ordering, meets and joins.

A-4.1 Transitivity

```
(cl:module (transitive)
        (forall (x y z)
                (iff    (and    (transitive x y) (transitive y z))
                        (transitive x z))
        )
)
```

A-4.2 Reflexivity

```
(cl:module (reflexive)
        (forall (x) (reflexive x x)))
```

A-4.3 Anti-Symmetry

```
(cl:module (anti-symmetry)
        (forall (x y)   (if     (and (asymm x y) (asymm y x))
```

```
                    (= x y)))
)
```

A-4.4 Associativity

```
(cl:module (associative)
        (forall (x y z)
                (=      (associative x (associative y z)) (associative (associative x y) z))
        )
)
```

A-4.5 Commutativity

```
(cl:module (commutative)
        (forall (x y)   (=      (commutative x y) (commutative y x))
))
```

A-4.6 Idempotency

```
(cl:module (idempotent)
        (forall (x)     (=      x       (idempotent x x))
))
```

A-5 Mapping Meet-Join to lte

This module establishes the connection between the meet-join axioms and less than or equal to. It should be noted that by extension, meet and join are then also mapped to maximal lower bound and minimal upper bound.

```
(cl:module (mj_to_lte)
(cl:import (poset))
(cl:import (meet))
```

```
(cl:import (join))
(cl:comment "This module defines a mapping to less than or equal to for meet and join")
        (forall (x y z)
                (iff    (= z (meet x y))
                        (and
                                (lte z x)
                                (lte z y)
                                (if     (exists (u)
                                        (and    (lte u x)
                                                (lte u y))
                                        (lte u z)
                                ))
        )))
         (forall (x y z)
                (iff    (= z (join x y))
                        (and
                                (lte x z)
                                (lte y z)
                                (if     (exists (u)
                                                (and    (lte x u)
                                                        (lte y u))
                                                (lte z u)
                                ))
        )))
)
```

Appendix B

Design Conceptualization Elements

B-1 Introduction

Welcome to MASULO.[18] This website is part of the larger CoLoRe initiative at the Semantic Technologies Laboratory at the University of Toronto. It is designed to serve as an integral resource for both ontology designers and users. The site is in continual upheaval driven by constant growth. Embracing a semi-open philosophy, anyone can become a contributor; otherwise they may simply use the tools that are offered. As an ontology resource it comprises of three components:

- A common logic ontology repository
- An ontology design tool
- A semantic mapping tool.

[18] Tentative title – MASULO – Metaphor a Source for Upper Level Ontology

The ontology repository is a collection of a diverse array of ontologies collecting and grouped in various layers of abstraction and concept similarity.

Building on top of the repository are two tools to help you realize the potential of ontological research. The first is an ontology design tool that utilizes the repository in conjunction with feedback from you to help you define a particular relation or function. The second is a semantic mapping tool that determines the *similarities* and *differences* between target ontologies of your choosing.

B-2 Ontology Design Tool

This is the ontology design tool component of this web service. Here, you may engage the site to help you define a relation or function without the burden of being an expert in formal logic. The basic philosophy behind this process is very simple.

Most people have difficulty being articulate in languages other than their mother tongues. Even if you are fluent in a learned language, for many it is difficult to express complex ideas clearly and precisely. This problem is particularly acute for formal logics as these languages often demand syntax and grammars that seem unwieldy and unnatural to many.

To address this problem, we have shifted the problem of communicating from one of writing in the language to going directly to the source. You will be engaging our service in a conversation based on models. Models here refer to particular arrangements of objects in the world that correspond to your intuitions about how the world works.

You will be communicating via pictures, diagrams or any of the other modalities of communication supported by our Sandbox Tool. For more information, click here to Learn More.

To begin your conversation, let's Get Started.

B-2.1 ODT Learn More

Ontology design tool implemented here attempts to circumvent the disconnect between subject matter experts and familiarity with formal logics. While many have difficulty expressing an idea easily, clearly and precisely in a foreign language, almost everyone knows whether an idea or concept they have is being used correctly or not.

The wonderful thing about formal logics is that they have defined with them a clear set of semantics, from which one can use to construct *models* of the concepts being defined. Models here correspond to particular objects which the concepts refer to, being arranged in a way that satisfies the semantics of the concepts.

For example, let us assume that you wanted to define the semantics for the *flow* relation. You know that *flow* is a binary relation, where something *flows* into something else. You could draw a picture, or model, showing how a number things flow into and out of one another.

You have extensionally defined semantics for *flow*. Unfortunately, it's not as easy as simply providing one, two or however many examples. The reason is that multiple semantics may give rise to the same models. Moreover, the models you drew may have accidental properties that you don't necessarily want invoked for all uses of *flow*.

Imagine you were asked to define the concept of *triangle* by examples. Most people tend to draw *isosceles* or *equilateral* triangles. While these triangles are certainly models of the concept *triangle*, if one were to construct a definition of triangle based on them, it would be too restrictive.

This is where the second component of our design tool kicks in. Making use of that shiny repository underpinning this entire website, we have constructed "maps" of logical

theories which help you weed out accidental properties and get to the axioms that you *really* want.

What's happening here is as follows:

1. We ask you to construct at least one (or more) models for the relation or function you are tying to define

2. Our software then converts your models into complete diagrams, which are a series of logical statements. Your models are then mapped into the repository to see which theories (definitions for you relation) give rise to your models.

3. Once this mapping has done, the initiative in the conversation is passed to our software agent.

4. The software agent will now use the repository to help you better define your relation.

5. It does so by now providing you with models that correspond to your relation in use.

6. All you have to do is decide whether the model is acceptable or not.

7. At the end of this process, the ontology design tool will provide you with the strongest theories in the repository that are consistent with all the models you deemed acceptable.

You might have noticed that we're still only giving you definitions based on models or examples. This means that there is always the possibility that there is some model out there that would invalidate the current definition. Does this sound familiar? We're embracing the scientific principle of falsification here. The definitions we give you are

only as good as the falsified hypothesis we've provided you. In fact, each of the models that our software generated was a hypothesis to see if a particular situation was applicable or not.

B-2.1.1 Limits of this approach

A quick word about some areas where the design tool might not be able to help you.

First, we're limited by the theories that are currently represented in this repository. We're really taking the reuse and modularity ideal of ontology research to heart here. Everything we use to help you define your concept or relation is a theory noted and articulated by someone, somewhere. As we've noted earlier, this repository is always growing, so if your models don't return any matches at the moment, check back soon!

The second limitation is that we really need you to know whether models make sense or not. While we're doing our best to provide you with the necessary tools to express your models in the most intuitive sense, if you can't decide whether a model is acceptable or not, the design tool won't be able to help you.

Thirdly, you might be trying to define too much. The relation you have in mind might actually behave different depending on what types of things it applies to. For example, the flow of electricity and the flow of water might have different properties. Confusing models for these two will cause our ontology design tool to recognize an ambiguous case, and terminate without providing axioms. What this might mean is that you actually have two or more distinct senses for your relation according to the *sorts* of arguments it ranges over. If you find our tool giving the sort error, pay closer attention to *what* your relation applies to, and see if there are actually more than one sense packed in there.

Lastly, there is the problem of showing models for particular types of relations. Specifically, visualizing (or audio-fying etc.) quarternary relations is fairly tricky. A 3-D model in motion / time is a common visualization, but it's not always appropriate.

Moreover, when constructing or looking at models which involve the notion of infinity, close attention must be paid to the convention used to represent the infinity.

B-2.1.2 Further Information

We hope the above has given you the adequate background to be able to comfortable use our ontology design tool. If you would like further information regarding the details of the algorithm and the tool, follow the links below to publications listed:

Hashemi, Ali. Master's Thesis 2008…
…. 2009 IJCAI paper
.....2009 CCAI paper
…. 2XXX Some Journal Paper

B2.2 Get Started

There are four ways to get started, please pick one of the following below:

Search Applications
You can either browse applications of theories in the repository and how they've been used, or relations that people have defined and which theories they map to.
Interactive Navigation
This is the ontology design tool and the algorithm, you draw some models, we'll map it into the repository and start providing you with model hypotheses.
Sandbox Tool
This is an environment to play around with models and see what works. You can start here, but we'll need a bit more info before we can start the ontology design process.
FOL Tutorial
Unsure whether you have the basics down? Take this quick tutorial to get the fundamentals and learn some basic terminology.

B-2.3 Provide Info

Before we begin, we need a bit of information regarding what you want axioms for. Please fill out the form below.

B-3 Semantic Mapping Tool

The semantic mapping tool really gets at the underlying metaphors which connect so many of our ideas to one another. Again, we stress that the kernel, the core that makes all this possible is the ontology repository. Moreover, we again also stress that the repository is a changing, growing entity. So long as new words, concepts and phenomena are observed and people decide they're worthy of defining, the repository should continue to grow.

The idea behind the semantic mapping tool arises from the roles metaphor play in our understanding. Conceptual (or structural) metaphors are those which map the underlying logical structures of one concept to another. The repository corresponds to a huge base of such logical substructures.

You may now input *your* ontology or ontologies into our semantic mapping tool to see which theories are being reused. You can see how:

- the ontologies map into the repository
- the target ontologies are *similar* to one another
- how in pairs, the target ontologies are *different* from one another

To being the semantic mapping process, input either the URI's of where the ontologies are located, or upload text files corresponding to the axioms. If your ontology is not in CLIF format, feel free to use one of our transcribers to convert the axioms into CLIF syntax.

If you'd like to learn more about the semantic mapping tool, click on *Learn More*.

B-3.1 SM Learn More

The semantic mapping algorithm is quite straightforward.

For each target ontology that is inputted, we construct an image of that ontology with respect to consistent theories in the repository. Our algorithm explores every core-hierarchy and constructs a set of all theories that were consistent with your ontology.

Having done this for every ontology, we can now present the *similarities* and *differences* between ontologies to you.

To see similarity, of the target ontologies you inputted, select 1 or more (you can select all). The tool will then take the intersection of all their consistent theories, pruning the *weaker* ones, yielding the strongest theories in the repository which they have in common. This is what is *similar* between the ontologies (or between a target ontology and the repository if only 1 was selected).

For example, one ontology might have defined the wheel of a car to include not just the tire, but also the axel, the hub cap etc. Another might have defined wheel to simply be the

tire. The intersection or the *similarity* between the two ontologies according to our repository would be *tire.*

Differences can only be shown in pairs. Select any two ontologies from those inputted, and see which theories they *do not* share. These divergences represent the differences between the ontologies.

B-3.1.1Limitations of this Approach

The semantic mapping tool does not establish equivalence between target ontologies. This is because the strongest connection we can make between ontologies, or an ontology and the repository is consistency. We can't make stronger claims because the semantic mapping tool can only reflect what's represented in the repository. A target ontology might have axioms which are not captured by theories in the repository.

B-3.1.2 Further Information

If you'd like to learn more about the semantic mapping tool, follow the links to the publications below.

Hashemi, Ali. Master's Thesis
IJCAI paper?
CCAI paper?
Journal paper?

Bibliography

(Asterias et al 2003) J. Atserias, L. Villarejo, G. Rigau, E. Agirre, J. Carroll, B. Magnini, P. Vossen, "The MEANING Multilingual Central Repository." In: *Proceedings of GWC 2004* pages 23–30. c Masaryk University, Brno, 2003

(Baader et al 2003) F. Baader, I. Horrocks and U. Sattler, "Description logics as ontology languages for the semantic web." *Festschrift in honor of Jörg Siekmann, Lecture Notes in Artificial Intelligence*, pages 228-248, Springer-Verlag, 2003.

(Bao et al 2006) J. Bao, D. Caragea, and V. Honavar, "Modular Ontologies - A Formal Investigation of Semantics and Expressivity." In: *Proceedings of the First Asian Semantic Web Conference,* Beijing, China, Springer-Verlag. Vol. Vol. 4185, pp. 616-631, 2006.

(Bao & Honavar, 2006) J. Bao and VG. Honavar, "Divide and Conquer Semantic Web with Modular Ontologies - A Brief Review of Modular Ontology Language Formalisms" In: *WoMo 2006. First International Workshop on Modular Ontologies* 2006.

(Barker et al 2001) K. Barker, P. Clark, B. Porter, "A library of generic concepts for composing knowledge bases." In: *The First International Conference on Knowledge Capture (K-Cap'01)*, 2001.

(Bioportal 2008) The National Centre for Biomedical Ontology. "Bioportal" [Online]. Available: http://www.bioontology.org/tools/portal/bioportal.html [Accessed: November 25 2008].

(Brachman & Schmolze 1985) RJ. Brachman, and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science* vol. 9 iss. 2, 1985.

(Choi et al 2006) N, Choi, I-Y. Song, and H. Han, "A Survey of Ontology Mapping." *ACM SIGMOD Record* vol. 35, iss. 3 pp. 34 – 41, September 2006.

(Clark et al 2000) P. Clark, J. Thompson, B. Porter, "Knowledge patterns." In: Cohn, A., Giunchiglia, F., Selman, B. (Eds.), *The Seventh International Conference on Principles of Knowledge Representation and Reasoning(KR-2000)*, pp 591–600, 2000.

(CL Standard 2007) International Organization of Standards. "ISO/IEC 24707:2007 - Information technology - Common Logic (CL) - A framework for a family of logic-based languages," Geneva, Switzerland: International Organization of Standards, 2007.

(Crubezy et al 2005) M. Crubezy, M. Musen, N. Noy and T. Redmond et.al. "The Protege Ontology Editor and Knowl-edge Acquisition System," August 2005. [Online] Available: http://protege.stanford.edu/ [Accessed: November 27, 2008]

(Cui and O'Brien 2000) Z. Cui and P. O'Brien, "Domain Ontology Management Environment." In *Proceedings of the 33rd Hawaii International Conference on System Sciences* - pages 1-9, 2000.

(Dameron et al 2004) O. Dameron, NF. Noy, H. Knublauch, MA. Musen, "Accessing and manipulating ontologies using web services." In: Proc. Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications at the Third International Conference on the Semantic Web (ISWC), 2004. [Online] http://smi-web.stanford.edu/people/noy/papers/dameron2004iswc.pdf. [Accessed November 26, 2008].

(Danesi 2002) M. Danesi, "Abstract Concept-Formation as Metaphorical Layering." *Studies in Communication Sciences 2/1* pages 1-22, 2002.

(Dewey 1938) J. Dewey, *Logic: The Theory of Inquiry*. Original: New York: Henry Holt & Co., 1938. New edition: Read Books, 2007.

(Ding & Fensel 2001) Y. Ding and D. Fensel, "Ontology Library Systems: The key to successful Ontology Re-use." In: *Proceedings of the First Semantic Web Working Symposium.* pages 93-112. Stanford University 2001.

(Doan et al 2002) A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to Map between Ontologies on the Semantic Web." In: Proceedings of the 11th International World Wide Web Conf. (WWW2002), 2002.

(Donnelly et al 2006) M. Donnelly, T. Bittner, and C. Rosse, "A formal theory for spatial representation and reasoning in biomedical ontologies." *AI in Medicine*. Vol. 36, Issue 1, pages 1-27, January 2006.

(Farquhar et al 1996) A. Farquhar, R. Fikes, and J. Rice, "The OntolinguaServer: a Tool for Collaborative Ontology Construction." In: *Proceedings of the Tenth Knowledge Acquisition forKnowledge-Based Systems Workshop,* 1996.

(Farrell & Lausen 2007) J. Farrell, and H. Lausen, "Semantic Annotations for WSDL and XML Schema" August 28, 2007. [Online] Available: http://www.w3.org/TR/sawsdl/ [Accessed: November 26, 2008]

(Fellbaum 1998) C. Fellbaum, *Wordnet: An Electronic Lexical Database.* Cambridge, MIT Press, 1998.

(Feynman 1969) R. Feynman, "What is Science?" *presented at the fifteenth annual meeting of the National Science Teachers Association, in New York City* (1966) published in *The Physics Teacher* Vol. 7, issue 6 (1969).

(Fikes & Farquhar 1997) R. Fikes and A. Farquhar, "Large-Scale Repositories of Highly Expressive Reusable Knowledge." Knowledge Systems Laboratory, Computer Science Department, Stanford University March 1997 [Online] ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-97-02.ps. [Accessed November 26, 2008].

(Fikes & Farquhar 1999) R. Fikes and A. Farquhar, "Distributed Repositories of Highly Expressive Reusable Ontologies" *IEEE Intelligent Systems*, vol.14 no. 2, pp.73-79, March 1999.

(Fluit et al 2003) C. Fluit, H. Horst, J. Meer, M. Sabou, and P. Mika, *'Spectacle', Towards the Semantic Web*, Wiley & Sons, 2003.

(Gangemi et al 1993) A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari, "Sweetening WordNet with DOLCE." *AI Magazine*, vol. 24 iss. 3, pages 13–24, 1993.

(Gangemi et al 2002) A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening Ontologies with DOLCE." In: *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management.*

Ontologies and the Semantic Web. EKAW 2002, pages 166-181 2002, Springer Verlag, 2002.

(Genesereth & Nilsson 1987) MR. Genesereth, and N. Nilsson, *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers: San Mateo, CA, 1987.

(Giunchiglia et al 2007) F. Giunchiglia, M. Yatskevich, and F. McNeill, "Structure preserving semantic matching." In: *Proceedings of the ISWC+ASWC International workshop on Ontology Matching (OM), pages 13–24,* 2007.

(Grenon et al 2003) P. Grenon, B. Smith, and L. Goldberg, "Biodynamic Ontology: Applying BFO in the Biomedical Domain." In: *Ontologies in Medicine: Proceedings of the Workshop on Medical Ontologies*. Amsterdam, IOS Press, 2003.

(Gruber 1993) T.R. Gruber, "A translation approach to portable ontologies." *Knowledge Acquisition*, vol. 5, iss. 2, pages199-220, 1993.

(Gruber 1995) TR. Gruber, "Toward principles for the design of ontologies used for knowledge sharing." *International Journal of Human-Computer Studies*, Special issue: the role of formal ontology in the information technology. Volume 43, Issue 5-6 pages 907 – 928, Nov/Dec 1995.

(Gruninger 1996) M. Gruninger, "Designing and Evaluating Generic Ontologies." In: *Proceedings of the 12th European Conference of Artificial Intelligence,* 1996.

(Gruninger 2003) M. Gruninger, "Ontology of a Process Specification Language." *Handbook of Ontologies and Information Systems,*S. Staab (ed.), Springer-Verlag, Berlin, 2003.

(Gruninger & Menzel 2003) M. Gruninger and C. Menzel, "The Process Specification Language: Principles and Aplications." *AI Magazine*, vol. 24, pages 63-74, 2003.

(Hayes 1996) P. Hayes, "A Catalog of Temporal Theories." Tech Report UIUC-BI-AI-96-01, 1996, University of Illinois.

(Hayes 2004) P. Hayes, "RDF Semantics" W3C Recommendation 10 February 2004. [Online] Available: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ [Accessed: November 26, 2008]

(Herre et al 2006) H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe, and H. Michalek, "General Formal Ontology (GFO): A Foundational Ontology Integrating Objects and Processes. Part I: Basic Principles." *Onto-Med Report 8*, Research Group Ontologies in Medicine, Institute of Medical Informatics, Statistics and Epidemiology, University of Leipzig, Leipzig, Germany. 2006.

(Hong et al 2006) M. Hong, J. Tang, and J. Li, "Semantic Annotation Using Horizontal and Vertical Contexts." In: *ASWC 2006*, LNCS 4185, pp. 58 – 64, 2006.

(Hou et al 2005) CJ. Hou, MA. Musen, NF. Noy, "EZPAL: environment for composing constraint axioms by instantiating templates." *International Journal of Human-Computer Studies*, vol., iss. 5, pp. 578 – 596, May 2005.

(Hovy 1998) E. Hovy, "Combining and standardizing large-scale, practical ontologies for machine translation and other uses." In: *Proceedings of 1st International Conference on Language Resources and Evaluation,* 1998.

(ISGCI 2008) "Information System on Graph Inclusions v2.0" April 8, 2008. [Online] Available: http://wwwteo.informatik.uni-rostock.de/isgci/index.html [Accessed: November 27, 2008]

(Kalfoglou & Schorlemmer 2003) Y. Kalfoglou and M. Schorlemmer, "Ontology Mapping: The State of the Art," *The Knowledge Engineering Review*, vol. 18, iss. 1 pages 1-31, 2003.

(Kremer 1998) R. Kremer, "Visual Languages for Knowledge Representation." In: *Eleventh Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, April 18-23, 1998.

(Lakoff & Johnson 2003) G. Lakoff, and M. Johnson, *Metaphors We Live By*. 2003 edition. Chicago: University of Chicago Press. 2003.

(Lenat & Guha 1990) D. Lenat and RV. Guha, "Building Large Knowledge-Based Systems: Representation and Inference in the Cyc project. Addison-Wesley, 1990.

(Lenat 1995) Doug Lenat, "CYC: A large-scale investment in knowledge infrastructure." *Communications of the ACM*, vol 38 pages 33-38, 1995.

(Madhavan et al 2002) J. Madhavan, PA. Bernstein, P. Domingos and AY. Halevy "Representing and Reasoning about Mappings Between Domain Models." In: *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 80-86. 2002.

(McCune 2003) W. McCune, "Otter 3.3 Reference Manual." *Tech. Memo ANL/MCS-TM-263*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, August 2003

(McGuinness & van Harmelen 2004) D. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview." W3C Recommendation February 10, 2004. [Online] Available: http://www.w3.org/TR/owl-features/ [Accessed: November 27, 2008]

(McLuhan 1964) M. McLuhan, *Understanding Media: the extensions of man.* Critical Edition, WT. Gordon (editor), Corte Madera: Gingko Press, 2003.

(Musen 1992) MA. Musen, "Dimensions of knowledge sharing and reuse." *Computers and Biomedical Research*, vol. 25, pages 435-467, 1992.

(Mutton & Golbeck 2003) P. Mutton and J. Golbeck, "Visualization of Semantic Metadata and Ontologies." In: *Proceedings of the Seventh International Conference on Information Visualization.* pages 300-305, 2003. ISBN:0-7695-1988-1

(Myers, 1990) B. Myers, "Taxonomies of Visual Programming and Program Visualization." *Journal of Visual Languages and Computing*, pages 97-123, 1990.

(Nardi & Brachman 2003) D. Nardi, and R. Brachman, "An introduction to description logics." *The description logic handbook: theory, implementation, and applications.* F. Baade eds. pages 1 – 40, Cambridge University Press 2003.

(Niles & Pease 2001) I. Niles, and A. Pease, "Towards a Standard Upper Ontology." In: *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Chris Welty and Barry Smith, eds, Ogunquit, Maine, October 17-19, 2001.

(Noy, 2004) NF. Noy, "Semantic Integration: A Survey Of Ontology-Based approaches." *ACM SIGMOD Record*. vol. 33, iss. 4 (December 2004).COLUMN: Special section on semantic integration. pages 65 – 70, 2004.

(Noy & Musen 2000) N. Noy, and M. Musen, "PROMPT: algorithm and tool for automated ontology merging and alignment." In: Proceedings of AAAI'00, pages 450–455, 2000.

(Noy et al 2004) N. Noy, D. Rubin and M. Musen, "Making Biomedical Ontologies and Ontology Repositories Work." *IEEE Intelligent Systems*, vol. 19, iss. 6, pages 78-81, 2004.

(Noy & Stuckenschmidt 2005) NF. Noy and H. Stuckenschmidt, "Ontology alignment: An annotated bibliography." In: *Semantic Interoperability and Integration*, Schloss Dagstuhl, 2005.

(OBO 2008) "Open Biomedical Ontologies." [Online] Available: http://www.obofoundry.org/about.shtml [Accessed: November 27, 2008]

(Ontolog 2008) "Open Ontology Repository." [Online] Available: http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository [Accessed: November 27, 2008]

(Pan et al 2003) J. Pan, S. Cranefield and D. Carter, "A Lightweight Ontology Repository." In: *Proceedings of AAMAS'03*, July 14–18, 2003 p 632-638, 2003.

(Patel-Schneider et al 2004) PF. Patel-Schneider, P. Hayes, and I. Horrocks, "OWL Web Ontology Language Semantics and Abstract Syntax." W3C Recommendation 10 February 2004. [Online] Available: http://www.w3.org/TR/owl-semantics/ [Accessed: November 26, 2008]

(Petre & Green 1993) M. Petre, TR. Green, "Learning to Read Graphics: Some Evidence that 'Seeing' an Information Display is an Acquired Skill." *Journal of Visual Languages and Computing,* vol. 4, iss. 1, pages 55-70, 1993.

(Pietriga et al 2006) E. Pietriga, C. Bizer, D. Karger, R. Lee, "Fresnel: A Browser-Independent Presentation Vocabulary for RDF." In: *Lecture Notes in Computer Science (LNCS 4273), Proceedings of the 5th International Semantic Web Conference (ISWC 2006),* pages 158-171, November 2006.

(Pinker 2007) Pinker, Steven, *The Stuff of Thought.* New York: Viking. 2007.

(Pinto et al 1999) H. Pinto, A. Prez and J. Martins, "Some Issues on Ontology Integration." In: *[18] IJCAI-99 workshop on Ontologies and Problem-Solving Methods*, 7-1 - 7-12, 1999.

(PSL 2007) "PSL Ontology." January 15, 2007. [Online] Available: http://www.mel.nist.gov/psl/ontology.html [Accessed: November 27, 2008]

(Riazanov & Voronkov 2001) A. Riazanov and A. Voronkov, "Vampire 1.1." L*ecture Notes in Computer Science,* vol. 2083, Springer Berlin / Heidelberg, 2001.

(Roberts 1973) D. Roberts, *The Existential Graphs of Charles S. Peirce.* The Hague, Mouton & Co. N. V, 1973.

(Romain & Cox 1993) GC. Romain, and K. Cox, "A Taxonomy of Program Visualization Systems." *IEEE Computer* vol. 26, pages 11-24, 1993.

(Sapir 1929) E. Sapir, "The Status of Linguistics as a Science." In: *E. Sapir (1958): Culture, Language and Personality (ed. D. G. Mandelbaum).* Berkeley, CA: University of California Press, 1958.

(Schulz 2004) S. Schulz, "System Abstract: E 0.81." In: *Proceedings of 2nd IJCAR, LNAI 3097*, pages 223-228,Springer, 2004.

(Shvaiko 2005) P. Shvaiko, "A survey of schema-based matching approaches." *Journal on Data Semantics,* vol. 4, pages 146-171, 2005.

(Simmhan et al 2005) Y. Simmhan, B. Plale and D. Gannon, “A survey of data provenance in e-science.” *SIGMOD Record*, vol. 34, pages 31-36, 2005.

(Smith 1977) D.C. Smith, *Pygmalion: A Computer Program to Model and Stimulate Creative Thought.* Stuttgart, Basel, 1977.

(Smith et al 2007) B. Smith, M. Ashburner, C. Rosse, J. Bard J, W. Bug, W. Ceusters, et al, "The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration." Nature Biotechnology, vol. 25, iss. 11, pages 1251-1255, 2007.

(Sowa 2007) JF. Sowa, “Fads and Fallacies about Logic.” *Intelligent Systems, IEEE,* vol. 22, iss.2, pages 84-87, March-April 2007.

(Staab et al 2001) S. Staab, M. Erdmann, A. Mäedche, “Engineering ontologies using semantic patterns.” In: Preece, A., O’Leary, D. (Eds.), *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence(IJCAI-01), Workshop on E-Business & the Intelligent Web*, 2001.

(Staab & Maedche 2000) S. Staab and A. Maedche, “Ontology Engineering beyond the Modeling of Concepts and Relations.” In: *14th European Conference on Artificial Intelligence; Workshop on Applications of Ontologies and Problem-Solving Methods*. 2000.

(Storey et al 2001) MA. Storey, M. Musen, J. Silva, N. Ernst, R. Fergerson and N. Noy, “Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition” In: *Protégé. Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, 2001.

(Stumme & Maedche 2001) G. Stumme and A. Maedche, "FCA-MERGE: Bottom-Up Merging of Ontologies", In: *Proceedings of IJCAI 2001*, pages 225-234, 2001.

(SUMO 2004) Deborah Nichols, "Ontology of Geography." July 2004. [Online] Available: http://sigmakee.cvs.sourceforge.net/*checkout*/sigmakee/KBs/Geography.kif [Accessed: November 27, 2008]

(Tu et al 2005) K. Tu, M. Xiong, L. Zhang, H. Zhu, J. Zhang, and Y. Yu, "Towards imaging largescale ontologies for quick understanding and analysis." In: *Proceedings of ISWC 2005*. pages 702–715, 2005.

(Uschold & Gruninger 1996) M. Uschold and M. Gruninger. "Ontologies: Principles, methods and applications." *Knowledge Engineering Review* vol. 11, pages 93-196, 1996.

(van der Vlist 2001) E. van der Vlist, "Comparing XML Schema Languages." December 12, 2001. [Online] Available: http://www.xml.com/pub/a/2001/12/12/schemacompare.html [Accessed: November 26, 2008]

(Wache et al 2001) H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner, "Ontology-based integration of information - a survey of existing approaches." In: *Proceedings of the workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 108–117, 2001.

(Whorf 1940) B.L. Whorf, "Science and Linguistics." *Language, Thought and Reality* (ed. J. B. Carroll). Cambridge, MA: MIT Press, 1956.

(Xiong et al 2006) M. Xiong, YF. Chen, H. Zheng, and Y. Yu, "Towards Quick Understanding and Analysis of Large-Scale Ontologies." In: *ASWC 2006*, LNCS 4185, pp. 84–98, 2006