Contributions to ontology reasoning and modeling

by

Aaron Eberhart

B.A., University of Pittsburgh, 2013

B.S., Wright State University, 2017

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2024

Abstract

This dissertation comprises research in diverse areas related to both ontology reasoning and ontology modeling. While these two topics differ in certain important ways, fundamentally data representation and modeling is central to the development of common-sense and useful reasoning methods, and useful reasoning applications enhance and enrich semantic models with ontologies. The contributions to both topics in this document are approached separately from a unified perspective, where ontology modeling and reasoning do not merely exist as adjacent topics, but instead inform each other. The five internal chapters each correspond to unique and related publications that all orbit this core topic and are contextualized in the introduction with central themes, which are then re-examined in the conclusion.

Contributions to ontology reasoning and modeling

by

Aaron Eberhart

B.A., University of Pittsburgh, 2013

B.S., Wright State University, 2017

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2024

Approved by:

Major Professor
Pascal Hitzler

Copyright

Abstract

This dissertation comprises research in diverse areas related to both ontology reasoning and ontology modeling. While these two topics differ in certain important ways, fundamentally data representation and modeling is central to the development of common-sense and useful reasoning methods, and useful reasoning applications enhance and enrich semantic models with ontologies. The contributions to both topics in this document are approached separately from a unified perspective, where ontology modeling and reasoning do not merely exist as adjacent topics, but instead inform each other. The five internal chapters each correspond to unique and related publications that all orbit this core topic and are contextualized in the introduction with central themes, which are then re-examined in the conclusion.

Contents

List of Figures

List of Tables

Acknowledgments

The publications featured in this document are based upon work supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386 as well as work funded by the National Science Foundation under award number 2033521.

Dedication

For Felix

Chapter 1

Introduction

This cumulative dissertation comprises research in diverse areas related to both ontology reasoning and ontology modeling. While these two topics differ in certain important ways, fundamentally data representation and modeling is central to the development of common-sense and useful reasoning methods, and useful reasoning applications enhance and enrich semantic models with ontologies. In this chapter some initial context is provided from a high-level perspective so that readers may familiarize themselves with the general concepts behind the research presented in following chapters containing the academic publications. More specific information will be provided in the chapters themselves that pertains to the content involved. At the end of this chapter some motivation will be presented that provides some justification to bind the chapters together in a centralized way and that will be examined in the the conclusion.

1.1 Ontologies

The word *ontology* comes from a domain of philosophy, specifically metaphysics, that attempts to describe what things exist, and investigates the nature of this existence. Ontologies in computer science are called this way because they are used for a similar reason, though in this case the application is more pragmatic. An ontology is, to put it very briefly, a formal artifact that describes in a logically precise manner the way data should be organized in order to to represent the intended

Table 1.1: $\mathcal{EL}^+$ Semantics

Description	Expression	Semantics
Individual	a	$a \in \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Concept	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential Restriction	$\exists R.C$	$\{\ a \mid$ there is $b \in \Delta^{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\ \}$
Concept Subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Role Chain	$R_1 \circ \cdots \circ R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \cdots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$

with $\circ$ signifying standard binary composition

semantics of whatever domain it describes. This means that, like this dissertation, research into ontologies has both a formal part that investigates how they can be leveraged for reasoning applications, and also a user-centric part that involves the semantics of what people want to represent and how to facilitate this in an efficient way.

Ontologies are often expressed in the Web Ontology Language (OWL) [1], and OWL ontologies correspond to particular description logics (DL). Typically a DL has three major components all of which are used to form axioms with a flexible set of logical operators: classes, roles[1], and individuals. The type of description logic used can vary based on what logical operators are used in the ontology.

Typically OWL ontologies support description logics with operators such as conjunction $\sqcap$, disjunction $\sqcup$, existential quantification $\exists$, universal quantification $\forall$, and binary composition $\circ$, all of which are connected with $\sqsubseteq$ to form axioms out of expressions. The choice of which operators to support in a given ontology has significant impact on the semantics for both modeling and reasoning.

In Tables 1.1 and 1.2 we can see the semantics for the two different logics mentioned specifically in this document to illustrate this point. Table 1.1 shows the very simple logic of $\mathcal{EL}^+$ where

[1]Classes are sometimes referred to as concepts, and roles are sometimes referred to as properties. As the terms are equivalent we will prefer the ones shown here.

Table 1.2: $\mathcal{ALCH}$ Semantics

Description	Expression	Semantics
Individual	x	$x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Class	B	$B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Negation	$\neg B$	$\Delta^{I} \setminus B^{\mathcal{I}}$
Conjunction	$B \sqcap C$	$B^{\mathcal{I}} \cap C^{\mathcal{I}}$
Disjunction	$B \sqcup C$	$B^{\mathcal{I}} \cup C^{\mathcal{I}}$
Existential Restriction	$\exists R.B$	$\{ x \mid$ there is $y \in \Delta^{\mathcal{I}}$ such that $(x, y) \in R^{\mathcal{I}}$ and $y \in B^{\mathcal{I}} \}$
Universal Restriction	$\forall R.B$	$\{ x \mid$ for all $y \in \Delta^{\mathcal{I}}$ where $(x, y) \in R^{\mathcal{I}}$, we have $y \in B^{\mathcal{I}} \}$
Class Assertion	$B(a)$	$a^{\mathcal{I}} \in B^{\mathcal{I}}$
Role Assertion	$R(a, b)$	$(a, b) \in R^{\mathcal{I}}$
Negated Role Assertion	$\neg R(a, b)$	$(a, b) \notin R^{\mathcal{I}}$
Class Subsumption	$B \sqsubseteq C$	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
Class Equivalence	$B \equiv C$	$B^{\mathcal{I}} = C^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

there is no disjunction, negation, or universal quantification. This makes modeling and reasoning with $\mathcal{EL}^+$ ontologies quite a bit simpler than ontologies that use $\mathcal{ALCH}$ semantics shown in Table 1.2 where these connectives do appear and permit more complex expressions. This is not always a linear relationship when comparing description logics however, since we can see that $\mathcal{EL}^+$ does support $\circ$ while $\mathcal{ALCH}$ does not.

1.1.1 Ontology Reasoning

Ontology reasoning is a broad domain that takes the formal axioms and rules present in an ontology and derives new insights from them [2]. Ontology reasoning is usually deductive: it uses a set of known facts and the schema provided by the ontology with a reasoning algorithm to find out new information. This can be a fairly simple process like we see in part of Chapter 4 [3] where the description logic $\mathcal{EL}^+$ is used as input to the system, or it can be recursive and complex like the reasoning shown in Chapter 3 [4] with the more complicated logic called $\mathcal{ALCH}$. There are also

inductive approaches to ontology reasoning, which attempt to perform or augment reasoning with machine learning and neural network techniques as is shown in of Chapter 4 with long short-term memory networks. And of course the backbone of an efficient reasoning algorithm is often the data representation underpinning the algorithm, an example of which is presented Chapter 2 [5] that provides an interface between the reasoning application and the person who inputs the ontology according to their desired semantics.

1.1.2 Ontology Modeling

Ontology modeling addresses one of the other major aspects of ontology research, namely the semantics and methodology surrounding their development. This involves laying groundwork in software like we see in Chapter 2. But it also concerns aspects of usability and design preference as discussed in Chapter 5 [6], where an empirical study was undertaken to investigate how complex an ontology really needs to be in order to be useful. This is important for modeling as well as reasoning, since the necessary complexity of an ontology will determine a large portion of the reasoning complexity. In Chapter 6 [7] an example is presented of a particular ontology with examples that shows the results of modeling with a modular methodology that aligns with the observations presented in Chapter 5.

1.2 Motivation

Throughout this document there are examples that touch on various aspects of a unified research program. While the separate contributions themselves do not directly mention an overarching theme, it is not hard to detect some themes if a reader knows what to look for. For this reason a short set of related research questions are presented here before the subsequent chapters begin to focus and motivate them. Later in the conclusion in Chapter 7 the questions will be examined in depth.

1. How can software support both ontology development and modeling as well as reasoning?
2. What are novel ways in which ontology reasoning can be efficiently accomplished?
3. In what ways can ontology reasoning leverage other AI techniques?
4. How do the semantics of ontology modeling impact ontology reasoning?
5. What does an ontology that was developed with modeling techniques look like?

Figure 1.1: Motivating research questions

Each of these questions corresponds to one of the five chapters following this introduction, though all questions are probably relevant to some degree throughout. Chapter 2 addresses question 1, Chapter 3 investigates question 2, and so on.

1.3 Outline

This cumulative dissertation contains five chapters that each respond to one of the research questions shown in Figure 1.1. First Chapter 2 will present and discuss how an API can support ontology authoring and editing for modeling or reasoning. Next in Chapter 3 we will see how a reasoner can be developed with the same ontology API as the first chapter that provides an innovative reformulation of a traditional reasoner algorithm with unique benefits and design tradeoffs. The following Chapter 4 presents an alternative approach to ontology reasoning that tries to perform symbolic reasoning tasks with a neural network. Then in Chapter 5 we will see an evaluation of how ontology modeling could be done more simply with patterns, and in a way that could greatly impact the reasoning complexity of an ontology. Finally we will see an example of an ontology that was modeled using these patterns.

Each chapter begins with an overview of the publication and a short description of how it connects to the research presented here. Then a duplication of the camera-ready pre-print discussed in the chapter will be reproduced in the appendix for reference. At the end of the document

there will be a short conclusion that summarizes the topics discussed and revisits the motivating questions.

Chapter 2

A Functional API for OWL

2.1 Overview of Contribution

This chapter presents work that led to the development of an API for the Web Ontology Language (OWL) called (f OWL)[5]. The (f OWL) API is named in this way because it is implemented in Clojure, a dialect of the Lisp functional programming language, which uses parentheses to encapsulate functions as seen in the logo in Figure 2.1.

Figure 2.1: (f OWL) logo

(f OWL) provides an alternative style of interaction with ontologies to the commonly used OWL API [8] that is implemented in an object oriented style. The existence of alternatives like (f OWL) allows users and developers to interact with OWL ontologies with different styles of programming interfaces. The implementation style also leverages data structures and techniques unique to the language so that OWL ontologies can be interacted with in different ways that can yield positive results in terms of speed and efficiency for the computer and the user.

2.2 Connection to Research

(f OWL) is not by itself a major publication result, it does however connect directly to the subsequent paper where it was used to develop a reasoner. The primary contribution in this work besides its connection to the next chapter is the connecting role it shows between ontology reasoning and ontology modeling and development in response to research question 1. With the API a user can develop ontologies according to their preferred methodology and then later reason with them using applications that interface with the same API.

Chapter 3

Should I Stay or Should I Go: A New Reasoner for Description Logic

3.1 Overview of Contribution

The research discussed in the chapter regards a reasoner called Emi that was featured in [4]. The Emi reasoner is a reformulation of traditional tableau reasoners to solve axioms more like rule-based reasoners.

3.1.1 Tableau Reasoning Algorithms

A tableau algorithm [2] is an algorithm that can determine the consistency of an ontology. Usually ontologies that are expressed in logics that contain disjunction, conjunction, universal quantification, existential quantification, and negation, such as ALC and logics that have it as a sub-logic like $\mathcal{ALCH}$ in Figure 1.2, require tableau reasoners.

The tableau algorithm works by drawing out a graph of individuals in the ontology to determine if the membership of the individual can be verified for every class expression. To give an example, assume we have a very simple ontology such as the one shown in Figure 3.1[1].

Here there are two axioms and three assertions about them, and we can see how a tableau can

[1] In this example A,B,C are class names, R is a role name, and d,e are individuals.

$$\{A \sqsubseteq B,\ B \sqsubseteq \exists R.C,\ A(d),\ R(d,e),\ C(e)\}$$

Figure 3.1: Example Ontology Fragment

be constructed in Figure 3.2. Each of the two nodes corresponds to a unique individual from the ontology, and the label L contains a set of expressions that the individual is a member of. This simple example is easy enough to trace mentally by simply inspecting the axioms, but a larger ontology with more axioms and individuals can very easily produce a very large graph that is quite complex and interconnected.

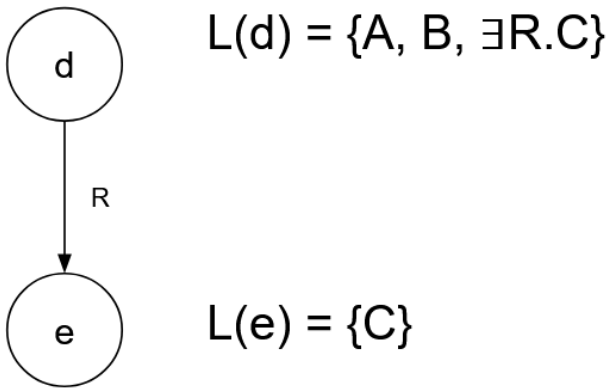


Figure 3.2: Example Tableau Graph

3.1.2 Rule-Based Reasoning Algorithms

Rule-based reasoners are designed for different types of logic such as Horn-logics and logic programming. These logics often have different semantics than description logics but can often work together when ontologies and rules have compatible expressivity.

A rule-based reasoner comparable to the tableau example seen previously is shown in Figure 3.3. Here we have the same two axioms and three assertions, they are just written as if they were logic programming axioms. If we want to see how a rule-based reasoner functions, we an simply take the axioms along with all of the assertions we know as facts, and then any time the antecedent of an axiom is true based on the facts that we have we also add the consequent of the axiom. For

{A(x) :- B(x), B(x) :- R(x,y) ∧ C(y), A(d), R(d,e), C(e)}

Figure 3.3: Example rules

example, because we know $A(x) : -B(x)$ and we know $A(d)$, then we can also add $B(d)$ to the facts that we know.

This type of reasoning can also be quite complex as well in more elaborate scenario. However there is a sense in which it can be more intuitive, since at any stage the axiom remains the same and a human often might solve the reasoning problem in a similar way by attempting substitution.

3.1.3 Emi

The Emi reasoner is an attempt to re-imagine the tableau algorithm as a complex type of rule reasoner. This is done by taking the tableau expansion rules such as shown in Figure 3.1 [2] that are used to solve the tableau algorithm and inverting them like we can see in Figure 3.2, so that instead of making a graph with individuals labelled by their expressions, the algorithm instead works with expressions labelled by their individuals. This minor change allows the Emi reasoner to 'solve' axioms as if they were rules by substitution as long as no backtracking is required.

		the following expansion rules apply to element x when x is not blocked
$\sqcap$-rule	if 1.	$(C \sqcap D) \in \mathcal{L}(x)$
	2.	$\{C, D\} \notin \mathcal{L}(x)$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$-rule	if 1.	$(C \sqcup D) \in \mathcal{L}(x)$
	2.	$\{C, D\} \cap \mathcal{L}(x) = \emptyset$
	then	either $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C\}$
		or else $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{D\}$
$\exists$-rule	if 1.	$\exists R.C \in \mathcal{L}(x)$
	2.	there is no y s.t. $\mathcal{L}((x, y)) = R$ and $C \in \mathcal{L}(x)$
	then	create a new node y and edge (x, y) with $\mathcal{L}(y) = \{C\}$ and $\mathcal{L}((x, y)) = R$
$\forall$-rule	if 1.	$\forall R.C \in \mathcal{L}(x)$
	2.	there is some y s.t. $\mathcal{L}((x, y)) = R$ and $C \notin \mathcal{L}(x)$
	then	$\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{C\}$

Table 3.1: $\mathcal{ALC}$ Tableau Expansion Rules

A side effect of this inversion of the tableau is the emergence of many new and potentially

$\sqcap$-rule	if 1.	$x \in \mathcal{L}(C \sqcap D)$
	2.	$x \notin \mathcal{L}(C) \cap \mathcal{L}(D)$
	then	$\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$
		$\mathcal{L}(D) \rightarrow \mathcal{L}(D) \cup \{x\}$
$\sqcup$-rule	if 1.	$x \in \mathcal{L}(C \sqcup D)$
	2.	$x \notin \mathcal{L}(C) \cup \mathcal{L}(D)$
	then	either $\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$
		or else $\mathcal{L}(D) \rightarrow \mathcal{L}(D) \cup \{x\}$
$\exists$-rule	if 1.	$x \in \mathcal{L}(\exists R.C)$
	2.	there is no y s.t. $(x, y) \in \mathcal{L}(R)$ and $y \in \mathcal{L}(C)$
	3.	there is no z s.t. x is blocked by z
	then	create a new element y with $y \in \mathcal{L}(C)$ and $(x, y) \in \mathcal{L}(R)$
$\forall$-rule	if 1.	$x \in \mathcal{L}(\forall R.C)$
	2.	there is some y s.t. $(x, y) \in \mathcal{L}(R)$ and $y \notin \mathcal{L}(C)$
	then	$\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$

Table 3.2: Inverted $\mathcal{ALC}$ Tableau Expansion Rules

powerful optimizations that may not translate to a traditional tableau. One such optimization, called ABox expansion, works lie the rule based reasoners, where a fact that we know for certain from the assertions can cause other new certain facts to emerge from the reasoning process that are a direct result of the assertions. Adding these to the reasoner allows it to vastly reduce the potential search space for a solution by removing impossible configurations from consideration. Another optimization emerges from the need to reformulate the notion of blocking in a functional programming style to ensure termination where there are no global references or variables, which we call local blocking. By maintaining a history where newly created individuals came from, and where their ancestors originated, it is quite straightforward to simply prevent axioms from creating cycles by stopping them from generating new individuals whenever an individual in their history exists as a result of the axiom that is currently being solved.

3.2 Connection to Research

The Emi reasoner and its accompanying publication demonstrate that there is indeed value in reexamining established reasoning algorithms. Although Emi does not (and likely cannot) challenge fundamentally the proven complexity of reasoning with OWL ontologies, by approaching

the problem in a new way it demonstrates that for research question 2 there are many possible novel methods still available for researchers to explore that can optimize reasoning performance and eplainability.

Chapter 4

Completion Reasoning Emulation for the Description Logic EL+

4.1 Overview of Contribution

This chapter concerns the publication [3] where a simpler description logic $\mathcal{EL}^+$ is used to generate input for training a neural network, which is then evaluated to see if it can be used to emulate the behavior or a reasoner by learning the sequences of its inferences.

The inferences that occur in an $\mathcal{EL}^+$ reasoner are similar in some respects to the rule-based reasoners described in Chapter 3. However, because $\mathcal{EL}^+$ is so simple it can actually be evaluated without inspecting any individuals and instead applying a series of completion rules like can be seen in Figure 4.1 on axioms themselves. These rules can be used by taking each and searching for axioms in the ontology that fit the pattern on the left side, and then whenever a match is found adding new inferred axiom to the ontology of the form on the right hand side and repeating the process. For $\mathcal{EL}^+$ ontlologies there are always a finite number of these entailments that can be generated, and at a certain stage of doing this reasoning every axiom can be checked by every rule and nothing new will emerge. This stopping point is known as completion, hence the name completion rules.

Because this reasoning process is deterministic and follows certain pre-defined patterns in

Table 4.1: $\mathcal{EL}^+$ Completion Rules

(1)	$A \sqsubseteq C$	$C \sqsubseteq D$		$\models A \sqsubseteq D$
(2)	$A \sqsubseteq C_1$	$A \sqsubseteq C_2$	$C_1 \sqcap C_2 \sqsubseteq D$	$\models A \sqsubseteq D$
(3)	$A \sqsubseteq C$	$C \sqsubseteq \exists R.D$		$\models A \sqsubseteq \exists R.D$
(4)	$A \sqsubseteq \exists R.B$	$B \sqsubseteq C$	$\exists R.C \sqsubseteq D$	$\models A \sqsubseteq D$
(5)	$A \sqsubseteq \exists S.D$	$S \sqsubseteq R$		$\models A \sqsubseteq \exists R.D$
(6)	$A \sqsubseteq \exists R_1.C$	$C \sqsubseteq \exists R_2.D$	$R_1 \circ R_2 \sqsubseteq R$	$\models A \sqsubseteq \exists R.D$

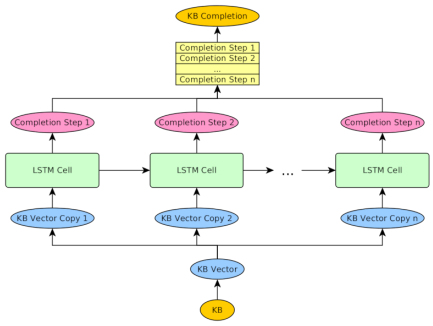


Figure 4.1: Flat Architecture

terms of the inferences it makes at each stage, the data collected from this reasoner was then fed into a series of different very simple Long Short-Term Memory (LSTM) networks and trained to trace the input ontology into the inferences that should emerge from reasoning with completion rules. The most basic control network can be seen in Figure 4.1 where no special help or assistance was given to the network and it was just naively trained as-is with the defined input and output sizes.

Two other networks were devised that roughly correspond to each other in order to test whether intermediate results from the reasoner can augment or improve the process. Figure 4.2 shows a network where the training is shaped according to this intermediate reasoning stage, but which is allowed to train and correct itself from the real inputs and outputs. In Figure 4.3 the same network is now split into two parts that are trained separately, first training the input layer to learn intermediate results, then training the output layer to learn answers from intermediate results. On evaluation results are fed from one network directly into the other so that both networks were comparable.

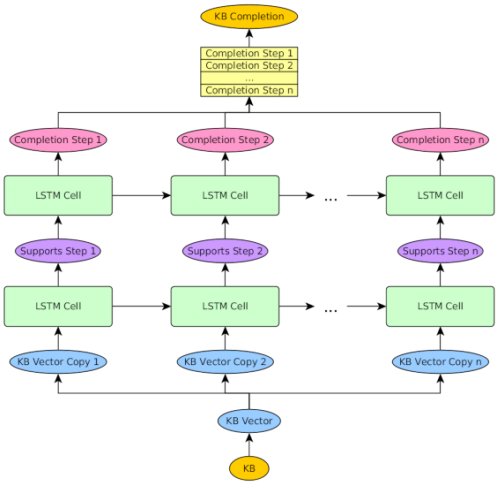


Figure 4.2: Deep Architecture

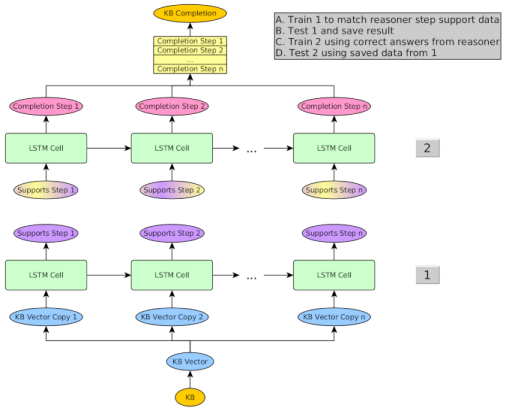


Figure 4.3: Piecewise Architecture

Although none of these LSTMs achieved groundbreaking accuracy, they did begin to very closely resemble the shape and patterns of the correct answers, even though the answers themselves were not exact.

4.2 Connection to Research

This chapter relates to the research question 3 by demonstrating how reasoners can be used with other AI techniques such as LSTMs and neural networks to investigate interesting new ways of performing inference and combining the unique benefits of different styles of AI. The potential to learn with intermediate answers could provide a way to peek inside the black box of many deep learning systems and gain insights into what they are really learning and how this can be leveraged for reasoning purposes.

Chapter 5

Expressibility of OWL Axioms with Patterns

5.1 Overview of Contribution

The publication [6] featured in this chapter takes a step away from reasoning directly and investigates how ontologies could be modeled, based on their semantics, and whether this could have been simpler if written differently. This paper is primarily concerning modeling, but reasoning still plays a role as complexity of axioms and how they are authored can impact the performance of a reasoner. From a modeling perspective the complexity of axioms is critical, however, as writing and working with axioms when modeling can have a steep learning curve.

Complexity is managed in the modeling process by adhering to a modular methodology [9] that uses patterns to formulate common simple axioms that authors often use when writing ontologies that can also be easily displayed in a visual editor. To evaluate whether existing ontologies *could* have been written with these simple patterns (even though they weren't), an empirical analysis was done to see what percentage of the axioms they contained might have emerged from pattern.

This involves first doing a very simple normalization that has no impact on the semantics of the ontology, in case some axioms were written in ways that could very easily be broken down into multiple simple parts. Next these axioms were matched against the axiom patterns shown in Table

5.1 in much the same way as the $\mathcal{EL}^+$ completion reasoner examines axioms, except in this case it is not doing inference but instead counting axioms that were or could have been expressed with each pattern.

Subclass	$A \sqsubseteq B$	Functional	$\top \sqsubseteq \leqslant 1R.\top$
Disjoint Classes	$A \sqcap B \sqsubseteq \bot$	Qualified Functional	$\top \sqsubseteq \leqslant 1R.B$
Domain	$\exists R.\top \sqsubseteq B$	Scoped Functional	$A \sqsubseteq \leqslant 1R.\top$
Scoped Domain	$\exists R.A \sqsubseteq B$	Qualified Scoped Functional	$A \sqsubseteq \leqslant 1R.B$
Range	$\top \sqsubseteq \forall R.B$	Inverse Functional	$\top \sqsubseteq \leqslant 1R^-.\top$
Scoped Range	$A \sqsubseteq \forall R.B$	Inverse Qualified Functional	$\top \sqsubseteq \leqslant 1R^-.B$
Existential	$A \sqsubseteq \exists R.B$	Inverse Scoped Functional	$A \sqsubseteq \leqslant 1R^-.\top$
Inverse Existential	$A \sqsubseteq \exists R^-.B$	Inverse Qualified Scoped Functional	$A \sqsubseteq \leqslant 1R^-.B$
		Structural Tautology	$A \sqsubseteq \geqslant 0R.B$

A,B,R are variable terms that contain at most one class or role name, or a data range

Table 5.1: OWLAx Axiom Patterns

In the resulting evaluation it turns out that, despite performing a very simple analysis, almost all of OWL axioms that occur in OWL ontologies that were inspected could have been or were simple and expressible with patterns.

5.2 Connection to Research

The content in this chapter addresses research question 4 by examining ways in which ontology development methodology that involves simple patterns of tractable axioms can inform and impact a reasoning algorithm. Because we can see that most of OWL doesn't need to be overly complex to still fulfil its purpose, reasoners like Emi that perform well on simple axioms can benefit from a unified design process that encompasses ontology development as well as ontology reasoning.

Chapter 6

A Domain Ontology for Task Instructions

6.1 Overview of Contribution

The final chapter in this document references a publication that uses the methodology [9] examined in the previous chapter and produces a tangible artifact. The ontology itself can be seen in Figure 6.1, as well as the modules that were used to construct it that are indicated by the grey boxes.

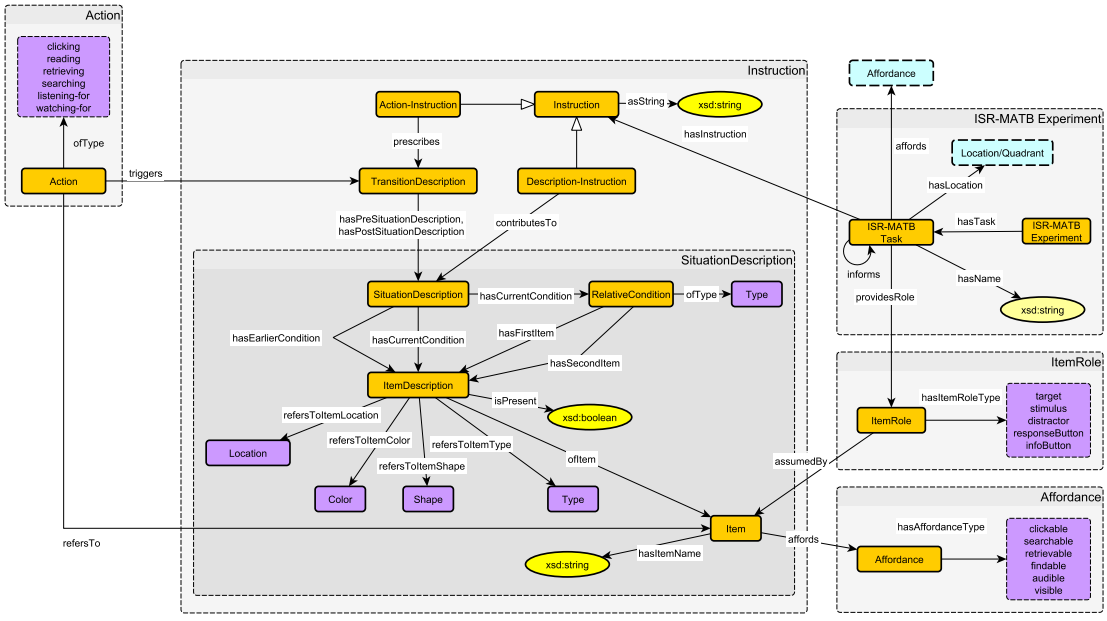


Figure 6.1: ISR-MATB Ontology

This ontology describes an experiment in cognitive science, where a cognitive agent was de-

signed to perform a set of tasks and respond as a human would. Each task involves some set of instructions, shown in the instruction module, that could describe the agent's environment or situation in the situation description. The agent could then take actions which might alter the environment or experiment. There are also modules to represent affordances the agent had, as well item roles for the items in the agent's environment and of course a module fr the experiment itself. Each axiom included in the ontology is described in detail in a short axiomatization for each module so that a reader can connect the high-level diagram with the formal axioms.

6.2 Connection to Research

This publication shows an example of what an ontology can look like when it is designed with the methodologies and techniques discussed in the previous chapter. It responds to research question 5 by providing such an example and describing the use case in detail to contextualize how the ontology development meets the needs of the users.

Chapter 7

Conclusion

To tie things together now here in the conclusion, the research questions from Figure 1.1 will be reexamined and discussed in context of their respective chapters.

Research Question 1:

How can software support both ontology development and modeling as well as reasoning?

In Chapter 2 an example was given of how software can be developed to represent the semantics of an OWL ontology in efficient ways to meet particular requirements of a user. With (f OWL), not only is there a flexible and lightweight API for ontology management in the functional programming paradigm, but also a framework upon which reasoning applications such as Emi can be developed. Although there are certainly other options for different use cases and preferences, (f

1. How can software support both ontology development and modeling as well as reasoning?
2. What are novel ways in which ontology reasoning can be efficiently accomplished?
3. In what ways can ontology reasoning leverage other AI techniques?
4. How do the semantics of ontology modeling impact ontology reasoning?
5. What does an ontology that was developed with modeling techniques look like?

Figure 1.1: Motivating research questions (repeated from page 5)

OWL) represents a strong example of how software can support both ontology development and ontology reasoning.

Research Question 2:

What are novel ways in which ontology reasoning can be efficiently accomplished?

The Emi reasoner presented in Chapter 3 demonstrates that new ways of approaching established algorithms an lead to unexpected benefits and insights. Although the Emi algorithm is theoretically related to existing algorithms in terms of complexity, due to the unique implementation it is able to achieve certain benefits that surpass other reasoners in efficiency. This is of course a design trade-off that benefits in some situations and adds complexity in others. In the end it is clear that efficient and novel ways to reason with ontologies are a potentially fruitful topic for further investigation.

Research Question 3:

In what ways can ontology reasoning leverage other AI techniques?

Chapter 4 presents a method whereby an LSTM neural network can be used to emulate a reasoner applying completion rules to an $\mathcal{EL}^+$ ontology. This method of reasoning trades the certainty of a deductive algorithm for the speed and flexibility of a trained neural network. While the range of possible AI topics that could augment ontology reasoning is quite broad, the research in this chapter provides some useful context into one such attempt to integrate and leverage other AI techniques in ontology reasoning.

Research Question 4:

How do the semantics of ontology modeling impact ontology reasoning?

The study presented in Chapter 5 investigates the extent to which the axiomatic complexity of OWL ontologies is inherent to the language or simply an artifact of design choices and authorship. To a surprising degree it turns out that ontologies are mostly very simple, even with a very naive

analysis, and could possibly even be simpler with a more thorough investigation. This hinges on the concept of expressibility, which relates to both ontology modeling and ontology reasoning. If we axioms are expressible simply with patterns, then doing so benefits both the modeler and the algorithm by removing ambiguity and complexity from the statements made.

Research Question 5:

What does a ontology that was developed with modeling techniques look like?

In Chapter 6 an example was provided that applied the axiom patterns that appeared in Chapter 5 to an actual use case. Here it can be seen how the rather simple statements made in the axiomatization lead to a robust ontology that can represent a range of cognitive science experiments. These simple axioms are used in a modular development methodology, whereby smaller modules are modeled independently and then put together to make a larger ontology. Modularization, in this sense, ensures that the ontology model is sufficiently precise while avoiding undesirable complexity in the modeling process as well as in any potential reasoning applications that may want to be applied in the future.

Bibliography

[1] Matthew Horridge and Peter F Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax. *W3C Working Group Note*, 2009.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, and Daniele Nardi. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[3] Aaron Eberhart, Monireh Ebrahimi, Lu Zhou, Cogan Shimizu, and Pascal Hitzler. Completion reasoning emulation for the description logic EL+. In Andreas Martin, Knut Hinkelmann, Hans-Georg Fill, Aurona Gerber, Doug Lenat, Reinhard Stolle, and Frank van Harmelen, editors, *Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice, AAAI-MAKE 2020, Palo Alto, CA, USA, March 23-25, 2020, Volume I*, volume 2600 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL https://ceur-ws.org/Vol-2600/paper5.pdf.

[4] Aaron Eberhart, Joseph Zalewski, and Pascal Hitzler. Should I stay or should I go - A new reasoner for description logic. In Fernando Ortiz-Rodríguez, Boris Villazón-Terrazas, Sanju Tiwari, and Carlos Bobed, editors, *Knowledge Graphs and Semantic Web - 5th Iberoamerican Conference and 4th Indo-American Conference, KGSWC 2023, Zaragoza, Spain, November 13-15, 2023, Proceedings*, volume 14382 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2023. doi: 10.1007/978-3-031-47745-4_2. URL https://doi.org/10.1007/978-3-031-47745-4_2.

[5] Aaron Eberhart and Pascal Hitzler. A functional API for OWL. In Kerry L. Taylor, Rafael S. Gonçalves, Freddy Lécué, and Jun Yan, editors, *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC)*, volume

2721 of *CEUR Workshop Proceedings*, pages 315–320. CEUR-WS.org, 2020. URL http://ceur-ws.org/Vol-2721/paper580.pdf.

[6] Aaron Eberhart, Cogan Shimizu, Sulogna Chowdhury, Md. Kamruzzaman Sarker, and Pascal Hitzler. Expressibility of OWL axioms with patterns. In Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Óscar Corcho, Petar Ristoski, and Mehwish Alam, editors, *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2021. doi: 10.1007/978-3-030-77385-4_14. URL https://doi.org/10.1007/978-3-030-77385-4_14.

[7] Aaron Eberhart, Cogan Shimizu, Christopher A. Stevens, Pascal Hitzler, Christopher W. Myers, and Benji Maruyama. A domain ontology for task instructions. In Boris Villazón-Terrazas, Fernando Ortiz-Rodríguez, Sanju Mishra Tiwari, and Shishir K. Shandilya, editors, *Knowledge Graphs and Semantic Web - Second Iberoamerican Conference and First Indo-American Conference, KGSWC 2020, Mérida, Mexico, November 26-27, 2020, Proceedings*, volume 1232 of *Communications in Computer and Information Science*, pages 1–13. Springer, 2020. doi: 10.1007/978-3-030-65384-2_1. URL https://doi.org/10.1007/978-3-030-65384-2_1.

[8] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions – Volume 529*, OWLED'09, page 49–58, Aachen, DEU, 2009. CEUR-WS.org.

[9] Cogan Shimizu. Towards a comprehensive modular ontology IDE and tool suite. In Sabrina Kirrane and Lalana Kagal, editors, *Proceedings of the Doctoral Consortium at ISWC 2018 co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th to12th, 2018*, volume 2181 of *CEUR Workshop Proceedings*, pages 65–72, 2018.

Chapter 8

Appendix: Referenced publications

A Functional API for OWL

Aaron Eberhart[0000−0003−3007−5460] and Pascal Hitzler[0000−0001−6192−3472]

Data Semantics Lab, Kansas State University, USA
{aaroneberhart, hitzler}@ksu.edu

Abstract. We present (f OWL), a minimalistic, functional programming style ontology editor that is based directly on the OWL 2 Structural Specification. (f OWL) is written from scratch, entirely in Clojure, having no other dependencies. Ontologies in (f OWL) are implemented as standalone and homogeneous data structures, which means that the same exact functions written for single axioms or expressions often work identically on any part of an ontology, even the entire ontology itself. The lazy functional style of Clojure also allows for intuitive and simple ontology creation and modification with a minimal memory footprint. All of this is possible without ever needing to use a single class, except of course in the Ontologies one creates!

1 Motivation

The semantic web community has widely adopted the OWL API [2] for ontology creation and development. Many researchers and programmers find this software very useful and enjoy its Object-Oriented style. Other APIs for OWL have also been written in imperative languages like Owlready [3] in Python. However, some developers prefer to program in functional languages and find that the imperative style APIs are more of a hindrance than a help when dealing with ontologies. There has been work towards providing functional style programming for OWL, like Tawny-OWL [4], but this is fundamentally dependent on the OWL API so it is cosmetically functional but not obviously compatible with many of the unique features and optimizations of the functional style. In order to support other styles of programming for ontologies we have developed (f OWL), a functional perspective on ontology development.

2 (f OWL)

As an original standalone piece of software, (f OWL) is not a wrapper for the OWL API. It starts again at the beginning and is truly functional from the ground up. (f OWL) was written with Clojure because it is a functional language that compiles with the Java Virtual Machine, and thus will have a broad compatibility with any machine that can run Java. Clojure has the added benefit

Functional	Lisp-like syntax
No dependencies	Requires only Clojure
No classes	Objects can be created and manipulated directly
Ontology data structure	Homogeneous and uniform: can be traversed
Lightweight	Supports OWL 2, no unnecessary baggage
Clojure Efficiency	Lazy functions and optimizable recursion
Concise function names	Avoids verbose unhelpful Java conventions

Table 1. Features of (f OWL)

of intuitive native laziness, so writing efficient programs to create and manipulate ontologies is not a challenge. To ensure that the semantics are correct, (f OWL) directly implements the grammar of the OWL 2 Structural Specification [5]. And because it is entirely original, (f OWL) is divorced from any historical dependencies and commitments that clutter other software.

(f OWL) includes many novel optimizations that streamline ontology development. True to the functional paradigm (f OWL) uses functions and shuns classes. Indeed all OWL ontology objects can be created directly with (f OWL) functions. And since these functions are not bound up in an arbitrary class hierarchy, they can operate on independent data structures that are internally typed to represent OWL semantics. A (f OWL) ontology itself is simply another data structure made of collections of smaller similar structures. This means that in most cases an expression, an axiom, even an ontology can be traversed recursively as-is without writing more functions. (f OWL) also simplifies many of the unhelpful, highly verbose Java-style function names that occur in other programs and instead adopts a concise functional naming scheme.

(f OWL) is built and tested using Leiningen[1] and the source code can be found on GitHub.[2] During testing, it efficiently reads and writes copies of over 260 different functional syntax ontologies from various domains, including the massive Gene ontology [1], with no errors being detected. It is available on the Clojars[3] repository so that users can easily import it into their own Clojure projects. ClojureDoc[4] for (f OWL) is available, and documentation is also accessible in the read–eval–print loop (REPL) for individual functions while writing programs.

2.1 Examples from the REPL

– Create an axiom

```
fowl.core=> (ont/implies (ont/exists "r" "a") "b")

SubClassOf(ObjectSomeValuesFrom(r a) b)
```

[1] https://leiningen.org/
[2] https://github.com/aaronEberhart/fOWL
[3] https://clojars.org/onto.aaroneberhart/fowl
[4] https://cljdoc.org/d/onto.aaroneberhart/fowl/0.0.1-SNAPSHOT/doc/readme

– Show the documentation for a function

```
fowl.core=> (doc ont/makeOWLFile)

-------------------------
ontology.core/makeOWLFile
([ontology filename & fileType])
  Writes an owl file of the ontology with the supplied file name.
  Optional argument allows choice of file type. No option defaults to functional syntax.
  (Currently only functional syntax defined)
```

– Make an ontology with a Clojure threading expression

```
fowl.core=> (-> ont/emptyOntologyFile
                (ont/setOntologyIRI "http://www.test.iri")
                (ont/addAnnotations (ont/annotation "annotations" "are fun"))
                (ont/addPrefixes (ont/prefix "" "http://www.test.iri/")
                                 (ont/prefix "prefix" "http://www.prefix.iri/"))
                (ont/addAxioms (ont/implies "a" (ont/IRI "prefix" "b"))
                               (ont/implies (ont/or "d" "g") "a")
                               (ont/implies (ont/inverseRole "r") "s")
                               (ont/fact "a" "i")
                               (ont/fact "r" "j" "i")))

Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(prefix:=<http://www.prefix.iri/>)
Prefix(:=<http://www.test.iri/>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)

Ontology(<http://www.test.iri>

Annotation(:annotations "are fun")

ObjectPropertyAssertion(:r :j :i)
SubObjectPropertyOf(ObjectInverseOf(:r) :s)
SubClassOf(ObjectUnionOf(:d :g) :a)
SubClassOf(:a prefix:b)
ClassAssertion(:a :i)
)
```

– Add every third axiom from a vector into a set with a tail recursive loop

```
fowl.core=> (loop [counter 0
                   axiomSet #{}
                   axioms [(ont/implies (ont/exists "r" "a") "b")
                           (ont/implies (ont/or "b" "c") (ont/not (ont/or "d" "e")))
                           (ont/implies (ont/roleChain "r" (ont/inverseRole "s")) "t")
                           (ont/fact (ont/inverseRole "s") "i" "j")
                           (ont/fact "a" "i")
                           (ont/fact "d" "i" (ont/stringLiteral "l"))
                           (ont/implies (ont/roleChain "s" "q") "t")]]

              (if (empty? axioms)                 ; are there no axioms left?

               axiomSet                           ; return axiom set

               (recur                             ; else recurse with new values

                (inc counter)                     ; counter + 1

                (if (= 0 (mod counter 3))         ; is counter mod 3 == 0?
                 (conj axiomSet (first axioms))   ; add first axiom to set
                 axiomSet)                        ; else keep current axiom set

                (rest axioms))))                  ; remove first axiom

#{ObjectPropertyAssertion(ObjectInverseOf(s) i j)
  SubClassOf(ObjectSomeValuesFrom(r a) b)
  SubObjectPropertyOf(ObjectPropertyChain(s q) t)}
```

– Use a partial function and a list comprehension to make axioms for a class

```
fowl.core=> (let [aImplies (partial ont/implies "a")]
              (for [b ["c" (ont/all "r" "d") (ont/and "e" "f" "g")]] (aImplies b)))

(SubClassOf(a c)
 SubClassOf(a ObjectAllValuesFrom(r d))
 SubClassOf(a ObjectIntersectionOf(e f g)))
```

More examples are available on the project GitHub page.

3 Evaluation

As a preliminary benchmark, we choose a few operations in (f OWL) that are commonly used and have near equivalent counterparts in the OWL API: read an ontology, write an ontology, get the NNF of all class axioms. Table 2 shows the average time to complete a task, and Table 3 shows the average times when they are each scaled by the number of axioms in the ontology. As expected, the performance of (f OWL) with respect to the OWL API closely parallels the difference between native Java and Clojure. Though it is usually faster when reading files, as indicated by the scaled times, (f OWL) is slowed down while reading large files by the need to construct many immutable objects for the ontology, which is typical for Clojure programs. However, even a large (f OWL) ontology can be accessed and traversed in a very efficient manner. This allows (f OWL) to write files quickly. And it can get the NNF of all class axioms significantly faster than even optimized stream functions.

All testing was done on a computer running Ubuntu 20.04.1 64-bit with an Intel Core i7-9700K CPU@3.60GHz x 8, 47.1 GiB DDR4, and a GeForce GTX 1060 6GB/PCIe/SSE2.

	Read File	Write File	Get NNF
OWL API	167.1696	91.61577	23.737692
(f OWL)	1275.384	68.87327	13.186410

Table 2. Average of times in milliseconds for tested ontologies

	Read File	Write File	Get NNF
OWL API	85.367181	8.82415379	1.8501445
(f OWL)	60.724610	6.01522460	0.6555225

Table 3. Average times in nanoseconds scaled by number of axioms

4 Conclusion

(f OWL) provides a lightweight functional alternative for OWL ontology development. It has numerous benefits that will doubtless increase as development finalizes and it matures. (f OWL) is highly practical for developers who prefer a functional language, it supports all of OWL 2, and provides unique functional methods for structuring and manipulating ontology data.

5 Future Work

There are two major efforts in progress to improve this project. The first is writing and testing new functions to allow the parsing of XML and RDF OWL files. This capability is integrated into the current project framework but it is incomplete and takes a fair amount of time to test and debug. Additionally, because OWL files are frequently in XML and RDF format we have delayed the implementation of ontology imports until after these file types can be read. Once the imports are implemented (f OWL) should support at least as much semantics for standard OWL ontology development as other APIs. A custom reasoner is also in the early stages of development for (f OWL). We hope that this will streamline reasoning over (f OWL) ontologies by running while they are created.

Acknowledgement This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386.

References

1. Gene Ontology Consortium: The Gene Ontology (GO) database and informatics resource. Nucleic Acids Research **32**(Database-Issue), 258–261 (2004)
2. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011)
3. Lamy, J.B.: Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. Artificial intelligence in medicine **80**, 11–28 (2017)
4. Lord, P.: The semantic web takes wing: Programming ontologies with tawny-owl. In: Rodriguez-Muro, M., Jupp, S., Srinivas, K. (eds.) Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED 2013) co-located with 10th Extended Semantic Web Conference (ESWC 2013), Montpellier, France, May 26-27, 2013. CEUR Workshop Proceedings, vol. 1080. CEUR-WS.org (2013), `http://ceur-ws.org/Vol-1080/owled2013\_16.pdf`
5. Parsia, B., Patel-Schneider, P., Motik, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C recommendation, W3C (Dec 2012), http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

Should I Stay or Should I Go
A New Reasoner for Description Logic

Aaron Eberhart[1,2], Joseph Zalewski[1], Pascal Hitzler[1]

[1] DaSe Lab, Kansas State University, Manhattan KS 66506, USA
[2] metaphacts GmbH, 69190 Walldorf, Germany
aaron.eberhart@gmail.com

Abstract. We present the Emi reasoner, based on a new interpretation of the tableau algorithm for reasoning with Description Logics with unique performance characteristics and specialized advantages. Emi turns the tableau inside out, solving the satisfiability problem by adding elements to expressions rather than adding expressions to element node labels. This strategy is inspired by decidable reasoning algorithms for Horn Logics and $\mathcal{EL}^{++}$ that run on a loop rather than recursive graph-based strategies used in a tableau reasoner. Because Emi solves the same problem there will be a simple correspondence with tableaux, yet it will feel very different during execution, since the problem is inverted. This inversion makes possible many unique and straightforward optimizations, such as paralellization of many parts of the reasoning task, concurrent ABox expansion, and localized blocking techniques. Each of these optimizations contains a design trade-off that allows Emi to perform extremely well in certain cases, such as instance retrieval, and not as well in others. Our initial evaluations show that even a naive and largely unoptimized implementation of Emi is performant with popular reasoners running on the JVM such as Hermit, Pellet, and jFact.

1 Introduction

Knowledge graph schema are complex artifacts that can be very useful, but are often difficult and expensive to produce and maintain. This is especially true when encoding them in OWL (the Web Ontology Language) as ontologies for data management or reasoning. The high expressivity of OWL is a boon, in that it makes it possible to describe complex relationships between classes, roles,[3] and individuals in an ontology. At the same time, however, this high expressivity is often an obstacle to its correct usage that can limit adoption, and can hamper any practical reasoning applications by adding complexity to the reasoning process.

To manage the complexity of OWL some prefer to accept hardness as it is and develop tools to manage or simplify the tricky parts. However, it occasionally is the case that problems appear difficult when they are actually quite intuitive when expressed differently. When this happens it can be helpful to start again

[3] We refer to properties as **roles**, unless a distinction is relevant, as this is the standard description logic term. These include both object properties and data properties.

from the beginning and re-imagine what is possible by trying something entirely different. This will of course not alter the fundamental proven computational complexity of any reasoning or modeling problem. But it can lead to algorithms and design patterns that are more easily understandable, and potentially uncover unique use cases and optimizations.

In this spirit we have used a new API for OWL called (f OWL) [2] for our experiment that can represent and better facilitate ontology data for the reasoning tasks we want it to perform. A custom reasoning algorithm called Emilia[4] (Emi) was implemented specifically to leverage the unique advantages of the new API, and with time the two will be able to seamlessly work within a single framework. The Emi reasoner will be able solve all of the same problems that current reasoners are able to solve, but its execution follows an iterative rule-like path rather than recursive tableau.

This type of experiment may seem equivalent and redundant to logicians and mathematicians, and looking at it only in a purely formal sense this can seem to be the case, however Emi works entirely differently from current systems and presents many opportunities for new research. The reasoner presented here is a prototype which demonstrates that the technique is valid and no less efficient than other comparable Java Virtual Machine (JVM) reasoners – the potential for new unique research directions and optimization techniques using this method will be the subject of future works. Initial testing is underway of Emi, and it is already performing competitively with other state-of-the-art systems in use today, and is particularly efficient with instance retrieval, despite being a naive prototype system written in a high level language with only the most basic and straightforward of optimizations.

2 $\mathcal{ALCH}$

The Emi algorithm currently supports $\mathcal{ALCH}$ reasoning, and the syntax and semantics of that logic are given below. Future extensions will likely expand expressivity for additional description logics.

2.1 Syntax

The signature Σ for the Description Logic $\mathcal{ALCH}$ is defined as $\Sigma = \langle N_I, N_C, N_R \rangle$ where:

- N_I is a set of individual element names.
- N_C is a set of class names that includes $\top$ and $\bot$.
- N_R is a set of role names.
- N_I, N_C, N_R are pairwise disjoint

Expressions in $\mathcal{ALCH}$ use the following grammar:

[4] **E**ager **M**aterializing and **I**terating **L**ogical **I**nference **A**lgorithm

$$\mathbf{R} ::= N_R$$
$$\mathbf{C} ::= N_C \mid \neg\mathbf{C} \mid \mathbf{C} \sqcap \mathbf{C} \mid \mathbf{C} \sqcup \mathbf{C} \mid \exists\mathbf{R}.\mathbf{C} \mid \forall\mathbf{R}.\mathbf{C}$$

2.2 Semantics

An interpretation $\mathcal{I} = \left(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\right)$ maps $N_I,\ N_C,\ N_R$ to elements, sets, and relations in $\Delta^{\mathcal{I}}$ with function $\cdot^{\mathcal{I}}$. An axiom A in $\mathcal{ALCH}$ is satisfiable if there is an interpretation where $\cdot^{\mathcal{I}}$ maps all elements, sets, and relations in A to $\Delta^{\mathcal{I}}$. An ontology O is a set of axioms formed from $\mathcal{ALCH}$ expressions, and is satisfiable if there is an interpretation that satisfies all axioms it contains, this interpretation being a model for O. $\cdot^{\mathcal{I}}$ is defined in Table 1 below.

Table 1. $\mathcal{ALCH}$ Semantics

Description	Expression	Semantics
Individual	x	$x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Class	B	$B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Negation	$\neg B$	$\Delta^{I} \setminus B^{\mathcal{I}}$
Conjunction	$B \sqcap C$	$B^{\mathcal{I}} \cap C^{\mathcal{I}}$
Disjunction	$B \sqcup C$	$B^{\mathcal{I}} \cup C^{\mathcal{I}}$
Existential Restriction	$\exists R.B$	$\{\ x \mid$ there is $y \in \Delta^{\mathcal{I}}$ such that $(x, y) \in R^{\mathcal{I}}$ and $y \in B^{\mathcal{I}}\ \}$
Universal Restriction	$\forall R.B$	$\{\ x \mid$ for all $y \in \Delta^{\mathcal{I}}$ where $(x, y) \in R^{\mathcal{I}}$, we have $y \in B^{\mathcal{I}}\ \}$
Class Assertion	$B(a)$	$a^{\mathcal{I}} \in B^{\mathcal{I}}$
Role Assertion	$R(a, b)$	$(a, b) \in R^{\mathcal{I}}$
Negated Role Assertion	$\neg R(a, b)$	$(a, b) \notin R^{\mathcal{I}}$
Class Subsumption	$B \sqsubseteq C$	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
Class Equivalence	$B \equiv C$	$B^{\mathcal{I}} = C^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

3 (f OWL)

We use (f OWL)[2] because it includes many novel optimizations that streamline ontology and reasoner development. True to the functional paradigm (f OWL) uses functions and shuns classes. Indeed all OWL ontology objects can be created directly with (f OWL) functions. And since these functions are not bound up in an arbitrary class hierarchy, they can operate on independent data structures that are internally typed to represent OWL semantics. A (f OWL) ontology

itself is simply another data structure made of collections of smaller similar structures. This means that in most cases an expression, an axiom, even an ontology can be traversed recursively as-is without writing more functions. The use of immutable data structures by (f OWL) in standard Clojure style also permits straightforward implementation of concurrent processes that operate on ontology data.

3.1 (f OWL) and Emi

Emi makes use of many features of (f OWL), some of which are uniquely advantageous in that they have no comparable alternative in other reasoners or APIs. One major advantage with using (f OWL) is the ontology-as-data-structure approach, which means that once (f OWL) has loaded the ontology into memory, Emi can efficiently traverse and manipulate expressions in the ontology without worrying about concurrency, since a (f OWL) ontology is immutable. Once the ontology is processed and elements in the signature are assigned mutable memory for use during reasoning, it is straightforward to define a partially paralellizable reasoning process with Clojure built-in functions that can be precisely lazy or eager when needed. The optimal configuration for when to choose each strategy is subject to many design trade-offs and does not have an objective best answer, though we believe we have developed a decent strategy. Emi processes axioms eagerly, but sets of axioms lazily and in parallel when possible due to the general observation that many non-synthetic ontologies contain a large number of axioms, most of which are rather small. This can affect performance on synthetic datasets where the occurrence of large or complex axioms is potentially higher and a different strategy may work better.

4 Emi Reasoner

As mentioned in the introduction, the new reasoner we are developing works iteratively without a graph, and is more like reasoning in $\mathcal{EL}^{++}$ or datalog where the process runs on a loop that eventually terminates rather than an expansive graph traversal. To reason iteratively we effectively need to turn the tableau inside out so we can work directly with axioms instead of a graph of elements. This means we will need to look at each axiom individually and add elements to or remove elements from expressions according to the tableau expansion rules as if they were backwards. This process slowly builds up a model for the ontology by partially realizing expressions until they become satisfiable and no longer need to be modified. Since every element is inspected for every axiom and the inverted tableau rules are followed in expressions we know this will be consistent. We assume that all expressions in our algorithm will be converted into Negation Normal Form (NNF); it is important to emphasize that absolutely no other logical preprocessing is used in the algorithm. In this section we assume that readers are familiar with the basics of tableau reasoning, if not they can consult [1] for further information.

4.1 Partial Interpretations

This algorithm works by directly attempting to realize an ontology and eventually build a consistent interpretation using the axioms as they are written in NNF, by adding and removing elements until all axioms are satisfiable or the ontology is shown to be unsatisfiable. We refer to the changing states of a program as partial interpretations.

Before continuing to the definition, it is important to stop and emphasize the subtle difference between partial interpretations and standard interpretations. A partial interpretation *may* be equivalent to some interpretation, and indeed when Emi terminates and determines we have a satisfiable ontology this final partial interpretation is a model. However partial interpretations are not necessarily models and represent the states of the program as different assignments are attempted in order to find a model. This difference may be confusing at first for logicians who are used to only seeing standard interpretations, but the distinction is important not just for showing a simple proof but also for describing the actual implementation of the algorithm as well. Our terminology deliberately corresponds much more closely to how an actual implemented algorithm, rather than a theoretical proof of one, would function by avoiding whenever possible notions of infinity or concepts that would need to be represented by global variables that are pervasive in proofs and inconvenient or impossible to implement. We do this intentionally so that it can be understood by programmers as well as logicians; we hope not just to show correctness but that it is clear to anyone how they could actually write a reasoner such as this.

Definition 1. *A* partial interpretation *representing the current state of the algorithm when a function is called is* $\mathcal{I}^* = (\mathcal{R}^{\mathcal{I}^*}, \cdot^{\mathcal{I}^*})$ *where* $\cdot^{\mathcal{I}^*}$ *is a function that maps elements, sets, and relations in an ontology* O *to a realization* $\mathcal{R}^{\mathcal{I}^*}$ *of* O*.*

Definition 2. *A* realization *of ontology* O *is a set of assertions that states for every class name* A *and element* x *in* O *that* $x \in A^{\mathcal{I}^*}$ *or* $x \notin A^{\mathcal{I}^*}$ *and for every role name* R *and element pair* (x, y) *in* O *that* $(x, y) \in R^{\mathcal{I}^*}$ *or* $(x, y) \notin R^{\mathcal{I}^*}$*.*

Each partial interpretation in this algorithm corresponds to some realization with a function $\cdot^{\mathcal{I}^*}$. Note that there is no restriction against inconsistent realizations and partial interpretations, only that they must be complete. A partial interpretation $\mathcal{I}^*$ with realization $\mathcal{R}^{\mathcal{I}^*}$ of ontology O is said to be a model of O iff there is an interpretation $\mathcal{I}$ of $O \cup \mathcal{R}^{\mathcal{I}^*}$.

As mentioned previously, while $\mathcal{I}$ often denotes a single consistent interpretation, $\mathcal{I}^*$ denotes the *current interpretation* when a function is evaluated in the algorithm, and as such may change over time and contain information that is later proved to be incorrect. Frequently we will say things like "Add x to $E^{\mathcal{I}^*}$", and this indicates that the partial interpretation $\mathcal{I}^*$ will henceforth be modified in the program state to represent the described change to $\cdot^{\mathcal{I}^*}$ that produces the desired realization.

4.2 Inverted Tableau Expansion Rules

Inverting the tableau rules is straightforward when we turn the standard tableau terminology, such as occurs in [1,6], on its head and assume that "labels" represent the action of the $\cdot^{\mathcal{I}^*}$ function on class and role names in an ontology to produce a realization, and that for any class or role name E that $E^{\mathcal{I}^*} = \mathcal{L}(E)$. To connect this idea with labelling terminology in a graph we use a labelled object, which can represent both notions.

Definition 3. *A* labelled object E *is an object that is associated with some set* $\mathcal{L}(E)$*, or* label*.*

First we note that nodes and labels do not necessarily need to be connected in a graph to represent the semantics of $\mathcal{ALCH}$, and can be reinterpreted as freely associating labelled objects. A labelled object can be a node in a tableau graph or simply an isolated named object with no inherent graph connections. This means that a label for a complex expression can be defined recursively to match the semantics of a partial interpretation $\mathcal{I}^*$, e.g. when an expression $C \sqcup D$ is satisfiable after applying the inverted tableau rules we have $\mathcal{L}(C \sqcup D) = (C \sqcup D)^{\mathcal{I}^*} = C^{\mathcal{I}^*} \cup D^{\mathcal{I}^*}$. We can represent axioms equivalently as labelled objects without actually needing to make a graph as long as all names in the signature are not duplicated in expressions but actually reference the same labelled objects.

For this algorithm we consider the inverted $\mathcal{ALC}$ tableau rules sufficient for deciding $\mathcal{ALCH}$ satisfiability, since the addition of the $\mathcal{H}$ fragment does not permit complex role expressions and they do not require expansion to check. The RBox axioms that are not included here will be described in Table 4 with the TBox axioms. In Table 2 we show the tableau expansion rules for $\mathcal{ALC}$ [1] and assume the standard notion of blocking for them, which can be found with the original proofs, then show the inversions in Table 3 where blocking is defined in Subsection 4.4.

	the following expansion rules apply to element x when x is not blocked
$\sqcap$-rule	if 1. $(C \sqcap D) \in \mathcal{L}(x)$
	2. $\{C, D\} \notin \mathcal{L}(x)$
	then $\mathcal{L}(x) \to \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$-rule	if 1. $(C \sqcup D) \in \mathcal{L}(x)$
	2. $\{C, D\} \cap \mathcal{L}(x) = \emptyset$
	then either $\mathcal{L}(x) \to \mathcal{L}(x) \cup \{C\}$
	or else $\mathcal{L}(x) \to \mathcal{L}(x) \cup \{D\}$
$\exists$-rule	if 1. $\exists R.C \in \mathcal{L}(x)$
	2. there is no y s.t. $\mathcal{L}((x, y)) = R$ and $C \in \mathcal{L}(x)$
	then create a new node y and edge (x, y) with $\mathcal{L}(y) = \{C\}$ and $\mathcal{L}((x, y)) = R$
$\forall$-rule	if 1. $\forall R.C \in \mathcal{L}(x)$
	2. there is some y s.t. $\mathcal{L}((x, y)) = R$ and $C \notin \mathcal{L}(x)$
	then $\mathcal{L}(y) \to \mathcal{L}(y) \cup \{C\}$

Table 2. $\mathcal{ALC}$ Tableau Expansion Rules

$\sqcap$-rule	if 1.	$x \in \mathcal{L}(C \sqcap D)$
	2.	$x \notin \mathcal{L}(C) \cap \mathcal{L}(D)$
	then	$\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$
		$\mathcal{L}(D) \rightarrow \mathcal{L}(D) \cup \{x\}$
$\sqcup$-rule	if 1.	$x \in \mathcal{L}(C \sqcup D)$
	2.	$x \notin \mathcal{L}(C) \cup \mathcal{L}(D)$
	then	either $\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$
		or else $\mathcal{L}(D) \rightarrow \mathcal{L}(D) \cup \{x\}$
$\exists$-rule	if 1.	$x \in \mathcal{L}(\exists R.C)$
	2.	there is no y s.t. $(x, y) \in \mathcal{L}(R)$ and $y \in \mathcal{L}(C)$
	3.	there is no z s.t. x is blocked by z
	then	create a new element y with $y \in \mathcal{L}(C)$ and $(x, y) \in \mathcal{L}(R)$
$\forall$-rule	if 1.	$x \in \mathcal{L}(\forall R.C)$
	2.	there is some y s.t. $(x, y) \in \mathcal{L}(R)$ and $y \notin \mathcal{L}(C)$
	then	$\mathcal{L}(C) \rightarrow \mathcal{L}(C) \cup \{x\}$

Table 3. Inverted $\mathcal{ALC}$ Tableau Expansion Rules

The labelling terminology is used in this subsection to show tableau correspondence, though in general we avoid it since it is more obvious what is happening to non-logicians when we show our algorithm as an implementable process that produces interpretations, rather than a convoluted mathematical abstraction. Specifically, we believe it is more natural to think of a predicate that represents a set as a labelled object where the label represents the set associated with the predicate, than it is to invert this and represent all the elements as labels for connected sets of predicate names.

4.3 ABox Expansion

One of the more useful optimizations that this algorithm permits is the expansion of ABox axioms into many additional, unstated, expressions which must hold in every possible model. This expansion begins while the algorithm evaluates the ABox and detects any immediate clashes. Later while evaluating the TBox the expansion detects any clashes that cannot possibly be fixed to allow the algorithm to terminate quickly, and it can be done concurrently whenever the algorithm adds or removes elements from expressions. In Definition 4 we maintain the assumption that all expressions are NNF and use $\neg$ to simplify the appearance of equivalent expressions.

Definition 4. *For partial interpretation $\mathcal{I}^*$, expression E, and element/pair x:*

1. *A* clash *means $x \in E^{\mathcal{I}^*}$ and $x \in \neg E^{\mathcal{I}^*}$*
2. *x* must be in *E iff there is no $\mathcal{I}^*$ where modifying $\cdot^{\mathcal{I}^*}$ to obtain $x \in \neg E^{\mathcal{I}^*}$ does not cause a clash*
3. *x* must not be in *E iff there is no $\mathcal{I}^*$ where modifying $\cdot^{\mathcal{I}^*}$ to obtain $x \in E^{\mathcal{I}^*}$ does not cause a clash*
4. *An* unresolvable clash *means x must be in $E^{\mathcal{I}^*}$ and x must not be in $E^{\mathcal{I}^*}$*

The set of all elements that must be in an expression $E^{\mathcal{I}^*}$, denoted $E_m^{\mathcal{I}^*}$, is a subset of the elements of $E^{\mathcal{I}^*}$, and the set of all elements that must not be in $E^{\mathcal{I}^*}$, written $E_{\overline{m}}^{\mathcal{I}^*}$, is a subset of the elements of $\neg E^{\mathcal{I}^*}$. Like the expressions themselves, the exact members of $E_m^{\mathcal{I}^*}$ and $E_{\overline{m}}^{\mathcal{I}^*}$ are unknown at the beginning of the algorithm, so $E_m^{\mathcal{I}^*}$ and $E_{\overline{m}}^{\mathcal{I}^*}$ indicate the additional knowledge of constraints on satisfiability that grow as the algorithm runs and are thus referred to with the $\mathcal{I}^*$ as well. When we talk about element(s) being "added" to these sets (nothing is ever removed), it is to indicate the change in $\cdot^{\mathcal{I}^*}$ that must hold in all future $\mathcal{I}^*$. Sets of elements that must (not) be in a complex expression can be computed in a straightforward way from the components of the expression they are in, for example $(B \sqcap C)_m^{\mathcal{I}^*} \equiv B_m^{\mathcal{I}^*} \sqcap C_m^{\mathcal{I}^*}$ and $(B \sqcap C)_{\overline{m}}^{\mathcal{I}^*} \equiv B_{\overline{m}}^{\mathcal{I}^*} \sqcup C_{\overline{m}}^{\mathcal{I}^*}$ (note the use of DeMorgan for $\overline{m}$).

Expansion occurs during TBox evaluation by, whenever possible, propagating the elements that must (not) be in expressions from antecedent to consequent, e.g. if we have axioms $\{A(x), A \sqsubseteq B\}$ then $x \in A_m$, and when we check $A \sqsubseteq B$ it is often possible to conclude $x \in B_m$. This is not always possible for expressions containing disjunction and negation except in certain very specific cases. Regardless the effect is powerful. Maintaining these sets allows us to explore only potentially correct solutions by preemptively avoiding actions that will be inconsistent in every model and also detect unresolvable clashes so the evaluation can terminate more quickly.

4.4 Local Blocking

The notion of blocking is also required for termination due to the new elements[5] created by the existential function. Our notion of blocking corresponds to the usual definition in that it references the same cyclic patterns in roles, however Emi cannot use blocking in reference to the global role hierarchy which is not computed explicitly, so cycles are detected locally by tracing the dependency paths that emerge as new elements are created to satisfy expressions.

Definition 5. *A new element x was* created to satisfy *expression $\exists R.B$ in an algorithm A if there are partial interpretations $\mathcal{I}^*, \mathcal{I}^{*\prime}$ for A where for some y we have $(y, x) \in R^{\mathcal{I}^*}, x \in B^{\mathcal{I}^*}, y \in \exists R.B^{\mathcal{I}^*}$ and $y \notin \exists R.B^{\mathcal{I}^{*\prime}}$ if $x \notin \Delta^{\mathcal{I}^{*\prime}}$.*

Definition 6. *For expression $\exists R.B^{\mathcal{I}^*}$, an element x is* blocked *by element z if $(z, x) \in R^{\mathcal{I}^*}$ and x was created to satisfy $\exists R.B$, or if for $(z, y_0) \in R^{\mathcal{I}^*}$ where y_0 was created to satisfy $\exists R.B$ we have $n \geq 0$ pairs of elements such that $\bigcup_{k=1}^{n}(y_{k-1}, y_k) \cup \{(y_n, x)\} \subseteq \Delta^{\mathcal{I}^*} \times \Delta^{\mathcal{I}^*}$.*

Fortunately this restriction is rather intuitive to implement: we simply need the function for an existential to not create new elements when checking new elements that exist as a result of a prior application of the same function on

[5] When we say 'new element' this is equivalent to a fresh element symbol. It is written this way because we are avoiding terminology that refers to an infinite set of names and instead refer to what a program really does here, i.e. create something new

the same existential, since this will induce a cycle. It is effectively as if elements have a 'history' represented by the chain of pairs that they inherit from the element that created them and which also contains their own origin. An element will therefore not generate new elements in a function with the existential that created it, and not if the function for the existential created an element that it depends on for its existence, or that is in its 'history'.

4.5 Termination Condition

The final component necessary for the algorithm to work is a termination condition. Unlike a tableau, this algorithm will run on a loop and will actually attempt to directly realize the ontology. How then do we know that the algorithm has correctly realized all of the axioms? This is actually rather simple. If the algorithm begins an iteration of checking all axioms with initial state $\mathcal{I}_a^*$ and ends this iteration with state $\mathcal{I}_b^*$ without encountering any unresolvable clashes, then we know it can terminate successfully if $\mathcal{I}_a^* = \mathcal{I}_b^*$. Since we know that each $\mathcal{I}^*$ represents a possible interpretation, it is clear that $\mathcal{I}^*$ is a model when no clash occurs after every axiom is checked and all elements remain the same in all expressions.

Definition 7. *A partial interpretation $\mathcal{I}^*$ is said to* equal *partial interpretation $\mathcal{I}^{*\prime}$, or $\mathcal{I}^* = \mathcal{I}^{*\prime}$, iff $\forall E \in N_C \cup N_R$ we have $E^{\mathcal{I}^*} = E^{\mathcal{I}^{*\prime}}$.*

Equality can be verified by checking if the names in the signature have not changed any of their memberships. If an element has been added to the signature, or has moved into or out of any class or role, the algorithm must check the axioms one more time to see if these changes cause side effects. Otherwise the algorithm will have checked every axiom and found them to be satisfiable without making any changes and has produced a model.

4.6 Algorithm Definition

An outline of Emi is shown in Algorithm 1 that uses functions from Table 4.

Backtracking Explicit backtracking in this algorithm is handled by 'reporting' clashes and unresolvable clashes.

Definition 8. *A* report *for Algorithm 1 immediately terminates execution of whatever process is occurring and returns to the function that handles this report.*

A report is meant to act like a programming exception. For example, when an unresolvable clash is directly reported while evaluating the ABox the behavior is clear, the ontology is unsatisfiable so all reasoning stops and the algorithm immediately exits by returning False.

Standard clashes occur when the inverted tableau rules require an action that is known to be inconsistent, but which could potentially be resolved by making changes elsewhere. In this case the immediate action is to stop trying the

obviously inconsistent task and instead do something different. This type of clash is normally handled in Emi by the code that solves the inverted tableau rules in the same way that a standard tableau might. However, there are cases where the *sat* function must intervene, for instance when the antecedent of an axiom can neither lack nor contain an element and both assignments have been attempted to exhaustion. In this case we have indirectly found an unresolvable clash. If a clash is reported but is able to be resolved, the algorithm merely proceeds along as normal, and any problems that a removal causes will themselves be reported and dealt with in the same way. Emi internally ensures that backtracking does not return to an identical previous program state so that termination is not a concern. ABox expansion greatly impacts this functionality by automatically removing many impossible solutions from consideration, allowing these reports to happen faster.

Parallelization Because we are not working with one single expanded concept representing the entire set of axioms like a standard tableau, it is possible in this algorithm to parallelize many operations on separate axioms. In the simplest case, it is unproblematic for the algorithm to process more than one axiom at the same time so long as they do not share any class or role names. Clashes that occur as a result of two independent axioms will in either case still be detected elsewhere and are dealt with regardless of the ordering. Additionally, this idea can be extended in the implementation to allow for axioms that share names to be evaluated concurrently as well, so long as shared names are not modified in ways that can cause a clash and begin backtracking. Emi makes use of the simplest case already and the implementation of more complex parallelization is in development.

Table 4. *sat* function behavior for $\mathcal{ALCH}$ satisfiability of ontology O, class name A, class expressions B, C, and roles R, S

Axiom	Action
$A(a)$	If $a \in A^{\mathcal{I}^*}_{\overline{m}}$ report an unresolvable clash, otherwise add a to $A^{\mathcal{I}^*}$ and $A^{\mathcal{I}^*}_m$
$\neg A(a)$	If $a \in A^{\mathcal{I}^*}_m$ report an unresolvable clash, otherwise add a to $A^{\mathcal{I}^*}_{\overline{m}}$
$B(a)$	Add a new class A and the axiom $A \sqsubseteq B$ to O and replace $B(a)$ with $A(a)$
$R(a, b)$	If $(a, b) \in R^{\mathcal{I}^*}_{\overline{m}}$ report an unresolvable clash, otherwise add (a, b) to $R^{\mathcal{I}^*}$ and $R^{\mathcal{I}^*}_m$
$\neg R(a, b)$	If $(a, b) \in R^{\mathcal{I}^*}_m$ report an unresolvable clash, otherwise add (a, b) to $R^{\mathcal{I}^*}_{\overline{m}}$
$B \sqsubseteq C$	For all $x \in B^{\mathcal{I}^*}$ add x to C using the inverted tableau expansion rules. If a clash is reported, instead bactrack to remove x from B. If both report a clash, report an unresolvable clash.
$B \equiv C$	Do $sat(B \sqsubseteq C)$ and $sat(C \sqsubseteq B)$ and report any unresolvable clashes.
$R \sqsubseteq S$	For all $(x, y) \in R^{\mathcal{I}^*}$ add (x, y) to S. If a clash is reported, instead backtrack to remove (x, y) from R. If both report a clash, report an unresolvable clash.

Algorithm 1: $\mathcal{ALCH}$ satisfiability for ontology O

Result: A realization that is a model of O when $True$, otherwise $False$ when an unresolvable clash occurs

```
// Initialize an empty set to represent a realization for O
```
$\mathcal{I}^* \leftarrow \{\}$;

```
// Assume all E/E_m/E_m̄ are empty and add them to I*
```
for $E \in N_C \cup N_R$ **do**
  $E \leftarrow \{\}$;
  $E_m \leftarrow \{\}$;
  $E_{\overline{m}} \leftarrow \{\}$;
  $\mathcal{I}^* \leftarrow \mathcal{I}^* \cup \{E, E_m, E_{\overline{m}}\}$;

```
// Obtain NNF of all axioms
```
$\mathcal{F} \leftarrow$ NNF of ABox of O;
$\mathcal{A} \leftarrow$ NNF of TBox of O;

```
// Process ABox
```
for $F \in \mathcal{F}$ **do**
  $sat(F)$;
  if *an unresolvable clash is reported* **then**
    return *False*

```
// Process TBox
```
repeat
  $\mathcal{I}^*_{old} \leftarrow \mathcal{I}^*$;
  for $A \in \mathcal{A}$ **do**
    $sat(A)$;
    if *an unresolvable clash is reported* **then**
      return *False*
until $\mathcal{I}^* = \mathcal{I}^*_{old}$;
return *True*

4.7 Correctness

The correctness of this algorithm follows from the direct correspondence between the tableau rules and their inversions along with a few minor details. First we will examine the inverted tableau rules. Each rule contains two distinct states in both the standard and inverted rules, the antecedent 'if' clause where certain conditions must hold, and the consequent 'then' clause where changes are made to the state in order to find a model. Thus it will be sufficient to show that each 'if' and 'then' clause from the inverted tableau rules is re-writable into the corresponding clause in the standard tableau rules.

$\sqcap$-rule

'if' This state in the inverted tableau rules corresponds to the standard tableau rules, since in each case we know x should be in $C \sqcap D$, but it is either not in C or D or both.

'then' The change made in response to the 'if' clause is also equivalent, since the inversion adding x to $C^{\mathcal{I}^*}$ and $D^{\mathcal{I}^*}$ represents the notion that we now have $x \in C^{\mathcal{I}^*}$ and $x \in D^{\mathcal{I}^*}$ which is identical to the standard tableau rules adding $\{C, D\}$ to $\mathcal{L}(x)$.

$\sqcup$-rule

'if' This state in the inverted tableau rules corresponds to the standard tableau rules, since in each case we know x should be in $C \sqcup D$, but it is neither in C nor in D.

'then' The change made in response to the 'if' clause is also equivalent, since the inversion adding x to $C^{\mathcal{I}^*}$ or $D^{\mathcal{I}^*}$ represents the notion that we now have $x \in C^{\mathcal{I}^*}$ or $x \in D^{\mathcal{I}^*}$ which is identical to the standard tableau rules either adding $\{C\}$ to $\mathcal{L}(x)$ or adding $\{D\}$ to $\mathcal{L}(x)$.

$\exists$-rule

'if' This state in the inverted tableau rules corresponds to the standard tableau rules, since in each case we know x should be in $\exists R.C$, but the current state means that there is either no $(x, y) \in R$ such that $y \in C$ or there is no $y \in C$ such that $(x, y) \in R$. The blocking condition (3) only applies when x is blocked because it is a new individual that exists as a (possibly indirect) result of an element previously created to satisfy this expression. When x is blocked in this way it is clear that we have completed the first iteration of a cycle and do not need to continue.

'then' The change made in response to the 'if' clause is also equivalent, since the inversion creating some y such that $y \in C^{\mathcal{I}^*}$ and $(x, y) \in R^{\mathcal{I}^*}$ represents the notion that $x \in \exists R.C$, which is identical to the standard tableau rules creating a new node and edge for y with $\mathcal{L}(y) = \{C\}$ and $\mathcal{L}((x, y)) = R$.

$\forall$-rule

'if' This state in the inverted tableau rules corresponds to the standard tableau rules, since in each case we know x should be in $\forall R.C$, but the current state means that there is some y such that $(x, y) \in R$ and $y \notin C$.

'then' The change made in response to the 'if' clause is also equivalent, since the inversion adding every y to $C^{\mathcal{I}^*}$ wherever $(x, y) \in R^{\mathcal{I}^*}$ represents the notion that $x \in \forall R.C$, which is identical to the standard tableau rules adding $\{C\}$ to any y so that $\mathcal{L}(y) = \{C\}$.

The only remaining thing to discuss is that Algorithm 1 and the *sat* function are sound and complete. For this it is simple to outline without a lengthy proof, since as we mentioned previously, the sat functions will test all elements in all expressions with the tableau expansion rules, exactly as you would have in a tableau. It is unnecessary and in fact unhelpful in this algorithm to combine all expressions into a connected whole since each axiom is itself evaluated consistently.

The subsumption axioms are solved in such a way that they implicitly compute a class and role hierarchy, so any inference dependent on a hierarchy will

still be entailed. Next, the stop condition of Algorithm 1 when the interpretation has not changed is identical with a completed tableau that no longer requires expansion or backtracking since both are models. And of course all cases where unresolvable clashes can occur are sound because they indicate instances where an element simultaneously must be in and must not be in an expression so the algorithm should terminate. As for completeness, it is again clear that whenever an unresolvable clash occurs it is identical with a case when a tableau discovers a clash but cannot backtrack to correct it.

5 Evaluation

The implementation of the Emi reasoner is evaluated against other popular JVM reasoners such as Hermit[6], Pellet[7], and jFact[8] [4,7,8] using Leiningen[9], which can run Clojure as well as Java programs. 500 ontologies were randomly sampled, half from ODPs and Biomedical ontologies in [3], and half from the Ontology Reasoner Evaluation 2015 competition dataset. These files were altered by removing any axiom that is not expressible in $\mathcal{ALCH}$ for testing. Among the 500 ontologies, 56 of them either timed out on all reasoners or caused our evaluation program to crash in some unexpected way due to lack of heap. It is not entirely clear from our logs which reasoner may be breaking in this way so we omit them, though we are confident that Emi has no heap issues or memory leaks (it is quite difficult to even *intentionally* create memory leaks with Clojure) and it can load all files by itself without error.

All testing was done on a computer running Ubuntu 20.04.1 64-bit with an Intel Core i7-9700K CPU@3.60GHz x 8, 47.1 GiB DDR4, and a GeForce GTX 1060 6GB/PCIe/SSE2.

5.1 Results

Testing and development is ongoing for the Emi reasoner and these results are a first example of the potential for this type of algorithm. A preliminary example of the reasoning time can be found in Figure 1. In this example, 3 tests are performed and timed on each reasoner: initialization from a file in memory shown in Figure 2, satisfiability and consistency checking shown in Figure 3, and retrieval of the elements in all non-empty class and role names after reasoning has completed shown in Figure 4.[10] Each test was given a timeout of 5 minutes, and when a test timed out more than 5 tests in a row for a reasoner it was prevented from testing again on the same file to save time in the evaluation. Across all tests, when we compare the percent difference between the reasoner time and the average time for all reasoners we see that Emi is overall 9.8% faster than

[6] Hermit Version 1.4.5.519
[7] Openllet Version 2.6.5
[8] jFact Version 5.0.3
[9] `https://leiningen.org/`
[10] Raw data and charts are available for inspection `https://tinyurl.com/kgswc2023`

average, Hermit is 4.2% slower than average, Pellet is 42.3% faster than average, and jFact is 55.9% slower than average. No reasoner was ever more than 100% better than average or 300% worse than average. Emi, Hermit, and Pellet each had 2 files when they failed due to an exception, while jFact failed on 70 files, though this is largely due to an issue it has with anonymous individuals in the ORE dataset.

As you can see in Figures 2 and 3, the Emi reasoner has a relatively slow initialize time and time for satisfiability on small ontologies. This is partially due to the fact that Emi solves all ABox axioms without complex expressions while it loads the ontology. This allows it to preemptively reject any naively unsatisfiable ABox, and the cost of this is usually only a few milliseconds. Emi seems to scale better than other systems and its performance improves in comparison as the number of axioms increases. Also, Emi is definitively faster in every single test when asked to compute the elements of all non-empty classes and roles, except the two tests where it had a timeout when computing satisfiability. Emi finishes reasoning with this information already computed, it is in effect computing both things at once, and only needs a variable amount of time to answer in our tests because it has to sort out the anonymous elements from every expression for comparison with the other reasoners where these elements are hidden by the OWL API [5]. Otherwise it could answer this in constant time.

Looking more closely at the data, there are three large clusters at 163, 325, and approximately 5500 axioms due to the synthetic ontologies in the ORE dataset. As you can see, all reasoners appear to behave similarly across multiple files of the same size in the clusters. Except for these clusters there appears to be a mostly stochastic distribution in the performance of each reasoner across different files with jFact doing great on small files but not scaling well, while Hermit, Pellet, and Emi start off a bit slower and seem to scale much better on very large files.

An interesting pattern we have noticed in the evaluation is that Emi usually outperforms other reasoners when it checks large ontologies with empty or nearly-empty ABoxes. This case makes sense when you notice that in $\mathcal{ALCH}$, as long as all axioms do not contain either negation or Top in the antecedent, there will always be a model where every predicate is empty. This is the default state when Emi starts, so it simply checks everything and the initial realization turns out to be fine after the first iteration.

6 Future Work

In the future there is quite a bit of work left to finalize and sufficiently verify Emi. These improvements will be ongoing as part of any new extensions. Parallelization in particular will likely prove difficult to formally pin down, though empirically its use is not as yet seeming to be problematic when comparing against the behavior of other reasoners. Some obvious extensions that are planned for the near future are inverse roles, nominals, and cardinality expressions which can work quite directly with some of the already existing code. Role chains are

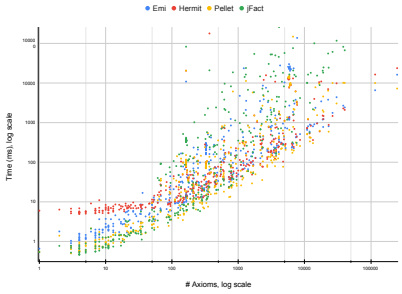


Fig. 1. All Reasoning Tests

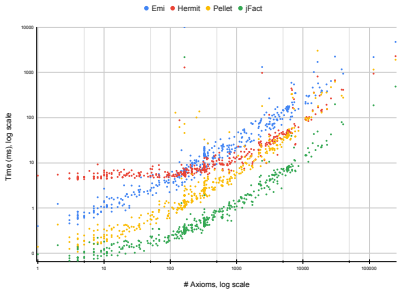


Fig. 2. Initialize

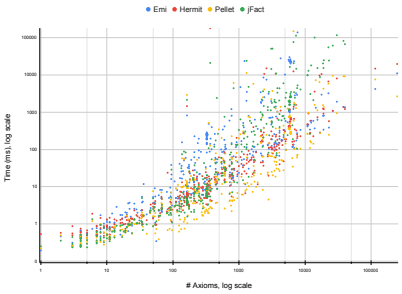


Fig. 3. Satisfiability

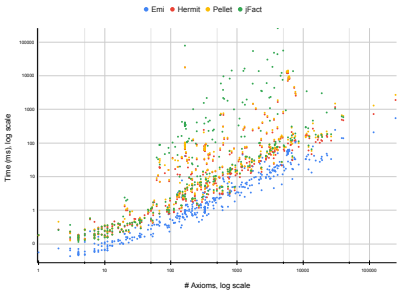


Fig. 4. Non-Empty Classes and Roles

another interesting and useful addition we are considering, though these will be difficult to implement efficiently so we will be cautious.

An interesting observation we have made while considering extensions is that some of the common difficulty with nominals in reasoning may turn out to be trivial in many cases for the Emi algorithm because nominals are not necessarily connected to anything as long as we maintain sufficient information about (in)equality. Once this is implemented and our hypothesis is checked it would also be straightforward to extend nominals to nominal schemas, since Emi does not normalize or pre-process away the original axioms. Initial testing suggest there is an intuitive way to bind nominal variables within axioms to solve this directly as Emi reasons.

Acknowledgement This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386.

References

1. Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., Nardi, D.: The description logic handbook: Theory, implementation and applications. Cambridge university press (2003)
2. Eberhart, A., Hitzler, P.: A functional API for OWL. In: Taylor, K.L., Gonçalves, R.S., Lécué, F., Yan, J. (eds.) Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th Inter-

national Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC). CEUR Workshop Proceedings, vol. 2721, pp. 315–320. CEUR-WS.org (2020), http://ceur-ws.org/Vol-2721/paper580.pdf
3. Eberhart, A., Shimizu, C., Chowdhury, S., Sarker, M.K., Hitzler, P.: Expressibility of owl axioms with patterns. In: Verborgh, R., Hose, K., Paulheim, H., Champin, P.A., Maleshkova, M., Corcho, O., Ristoski, P., Alam, M. (eds.) The Semantic Web. pp. 230–245. Springer International Publishing, Cham (2021)
4. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An owl 2 reasoner. J. Autom. Reason. **53**(3), 245–269 (oct 2014). https://doi.org/10.1007/s10817-014-9305-1, https://doi.org/10.1007/s10817-014-9305-1
5. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: Proceedings of the 6th International Conference on OWL: Experiences and Directions – Volume 529. p. 49–58. OWLED'09, CEUR-WS.org, Aachen, DEU (2009)
6. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible sroiq. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning. p. 57–67. KR'06, AAAI Press (2006)
7. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semant. **5**(2), 51–53 (jun 2007). https://doi.org/10.1016/j.websem.2007.03.004, https://doi.org/10.1016/j.websem.2007.03.004
8. Tsarkov, D., Horrocks, I.: Fact++ description logic reasoner: System description. In: Proceedings of the Third International Joint Conference on Automated Reasoning. p. 292–297. IJCAR'06, Springer-Verlag, Berlin, Heidelberg (2006). https://doi.org/10.1007/11814771_26, https://doi.org/10.1007/11814771_26

Completion Reasoning Emulation for the Description Logic $\mathcal{EL}^+$

Aaron Eberhart, Monireh Ebrahimi, Lu Zhou, Cogan Shimizu, and Pascal Hitzler
Data Semantics Lab
Kansas State University, USA
{aaroneberhart,monireh,luzhou,coganmshimizu,hitzler}@ksu.edu

Abstract

We present a new approach to integrating deep learning with knowledge-based systems that we believe shows promise. Our approach seeks to emulate reasoning structure, which can be inspected part-way through, rather than simply learning reasoner answers, which is typical in many of the black-box systems currently in use. We demonstrate that this idea is feasible by training a long short-term memory (LSTM) artificial neural network to learn $\mathcal{EL}^+$ reasoning patterns with two different data sets. We also show that this trained system is resistant to noise by corrupting a percentage of the test data and comparing the reasoner's and LSTM's predictions on corrupt data with correct answers.

Introduction

Machine learning and neural network techniques are often contrasted with logic-based systems as opposite in many regards. The challenge of integrating inductive learning strategies with deductive logic has no easy or obvious solution. Though there are doubtless many reasons for this, one major roadblock is that solutions and evaluations which work well for logic, or for deep learning and machine learning, do not work well in the opposite, and likely do not further the goal of integration. In this paper we present a new approach that embraces the liminality of this specific integration task. Our approach is by its very nature ill-suited to either machine learning or logic alone. But it tries to avoid the pitfalls of unintentionally favoring one paradigm over the other, aiming instead to grasp at something new in the space between.

Related Work

A popular approach for training neural networks to recognize logic is to use embedding techniques borrowed from the field of natural language processing (NLP). For instance, Makni and Hendler designed a system that layers Resource Description Framework (RDF) graphs into adjacency matrices and embedding them.(Makni and Hendler 2018) There is also considerable research into path-finding in embedded triple data using recurrent neural networks (RNNs) or LSTMs.(Xiong, Hoang, and Wang 2017; Das et al. 2016; 2017) Memory networks also successfully operate over triple embeddings and are capable of transferring training to work over triple embeddings from completely different vocabularies.(Ebrahimi et al. 2018) There is even an attempt to embed $\mathcal{EL}^{++}$ with TransE using a novel concept of n-balls, though it currently does not consider the RBox as well as certain $\mathcal{EL}^{++}$ axioms that do not translate into a triple format.(Kulmanov et al. 2019; Bordes et al. 2013)

Discussion

Many of the systems just mentioned are very successful, yet have a tendency to adopt familiar evaluation strategies from deep learning. Sure, it's great to be able to get a 99% F1-score. When we do not know the underlying structure of the data that we are using, this is often the best we can hope for. However, precision and recall do not make much sense in a knowledge base evaluation. For an integrated system, some new goals and evaluation strategies seem warranted that are neither completely deductive nor entirely empirical.

In a deductive reasoning system the semantics of the data is known *explicitly*. Why then would we want to blindly allow such a system to teach itself what is important when we already know what and how it should learn? Possibly we don't know the best ways to guide a complex network to the correct conclusions, but surely more, not less, transparency is needed for integrating logic and deep learning. Transparency often becomes difficult when we use extremely deep or complex networks that cannot be reduced to components. It also makes things difficult when we pre-process our data to help the system train by doing embeddings or other distance adjustments. When we embed all the similar things close together and all the dissimilar things far apart the system can appear to succeed, but we can't tell if it was the embedding or the system itself or one of a dozen other things that might have caused this.

In response to these and other concerns we have developed some research questions that we hope to address in this paper. Though we certainly will not be able to definitively answer all of them, we do hope that by starting this investigation we can demonstrate future potential for this type of inquiry.

Research Questions

1. Can a neural network emulate reasoner behavior?
2. Can we still achieve success without embeddings?
3. Can learning happen with arbitrary numerical names?
4. Will intermediate answers help our evaluation?
5. Is F1-score sufficient to evaluate an integrated logic and neural network system? If not, what would be?

Question one is the primary focus of this project, and the one to which all the others are subordinate. It is the intention of this paper and the prototype system we have implemented to demonstrate that this is a feasible goal. Whatever disagreements may arise concerning our later investigations, we will argue that this is a necessary and plausible alternative course of research.

The second question we have already mentioned in the last section. It is our intention to avoid any pre-processing that will allow the system to give correct answers without learning the reasoning patterns. In this way we can better verify whether we have truly solved the reasoning problem and not some other easier task.

The third question is related to the second, but imposes a more broad restriction on what we will be allowed to do. Not only do we want to avoid embedding the data in a way that will help the system to learn the answers. Also we must take care when we go from logic to numerical data, and back again, not to interfere with the latent semantics in the logic. By avoiding distance comparisons in the naming convention for the input we ensure that the names are arbitrary in both systems. The only distances we intend to measure are between predictions and answers.

The fourth question is very important for our conception of how future systems should work. In a logic-based system there is transparency at any level of reasoning. We cannot, of course, expect this in most neural networks. With a network that aims to emulate reasoner behavior, however, we can impose a degree of intermediate structure. This intermediate structure allows us to inspect a network part-way through and perform a sort of "debugging" since we know exactly what it should have learned at that point. This is a crucial break from current thinking that advocates more and deeper opaque hidden layers in networks that improve accuracy but detract from explainability. Inspection of intermediate answers could indicate whether a proposed architecture is actually learning what we intend, which should aid development of more correct systems.

The fifth and final question is a difficult but necessary question, likely beyond the scope of this paper. For now, it is primarily a reminder that the method we choose intentionally does not try to find best answers for individual statements. It tries to emulate reasoner behavior by matching the shape of the reasoning patterns. That is not to say we don't care about right answers, or that they don't matter. Those values are reported for our system. And if reasoning structure is matched well enough then correct answers should follow. But we believe that the reasoning distance results we report are far more compelling and tell a better story about what is happening, as well as indicate the future potential of this method.

Approach

We present a method for learning the structure of $\mathcal{EL}^+$ reasoning patterns using an LSTM that tests the questions posed above. In our system we avoid, as much as possible, embeddings or any distance adjustments that might help the system learn answers based on embedding similarity without first learning the reasoning structure. In fact, a primary feature of our network is its lack of complexity. We take a very simple logic, reason over knowledge bases in that logic, and then extract supports from the reasoning steps, mapping the reasoner supports back to sets of the original knowledge base axioms. This allows us to encode the input data in terms of only knowledge base statements. It also provides an intermediate answer that might improve results when provided to the system. This logic data is fed into three different LSTM architectures with identical input and output dimensionalities. One architecture, which we call "Deep", does not train with support data but has a hidden layer the same size as the supports we have defined. Another architecture, called "Piecewise", trains two separate half-systems, one with knowledge bases and one with supports from the reasoner. The last system, called "Flat", simply learns to map inputs directly to outputs for each reasoning step. We will discuss in detail each part of the system in the following sections, then provide some results.

$\mathcal{EL}^+$

$\mathcal{EL}^+$ is a lightweight and highly tractable description logic. A typical reasoning task in $\mathcal{EL}^+$ is a sequential process with a fixed endpoint, making it a perfect candidate for sequence learning. Unlike RDF, which reasons over instance data in triple format, $\mathcal{EL}^+$ reasoning occurs on the predicate level. Thus we will have to train the system to actually learn the reasoning patterns and logical structure of $\mathcal{EL}^+$ directly from encoded knowledge bases.

Syntax The signature Σ for $\mathcal{EL}^+$ is defined as:

$$\Sigma = \langle N_I, N_C, N_R \rangle \text{ with } N_I, N_C, N_R \text{ pairwise disjoint.}$$

N_I is a set of individual names.
N_C is a set of Concept names that includes $\top$.
N_R is a set of Role names.

Expressions in $\mathcal{EL}^+$ use the following grammar:

$$\begin{array}{lcl} \mathbf{R} & ::= & N_R \\ \mathbf{C} & ::= & N_C \quad | \quad \mathbf{C} \sqcap \mathbf{C} \quad | \quad \exists \mathbf{R}.\mathbf{C} \end{array}$$

Semantics An interpretation I where $I = (\Delta^I, \cdot^I)$ maps N_I, N_C, N_R to elements, sets, and relations in Δ^I with function $\cdot^I$. For an interpretation I and $C(i), R(i) \in \Sigma$, the function $\cdot^I$ is defined in Table 1.

Table 1: $\mathcal{EL}^+$ Semantics

Description	Expression	Semantics
Individual	a	$a \in \Delta^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Concept	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential Restriction	$\exists R.C$	$\{\ a \mid$ there is $b \in \Delta^{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\ \}$
Concept Subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Role Chain	$R_1 \circ \cdots \circ R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \cdots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$

with $\circ$ signifying standard binary composition

Table 2: $\mathcal{EL}^+$ Completion Rules

(1)	$A \sqsubseteq C$	$C \sqsubseteq D$		$\models A \sqsubseteq D$
(2)	$A \sqsubseteq C_1$	$A \sqsubseteq C_2$	$C_1 \sqcap C_2 \sqsubseteq D$	$\models A \sqsubseteq D$
(3)	$A \sqsubseteq C$	$C \sqsubseteq \exists R.D$		$\models A \sqsubseteq \exists R.D$
(4)	$A \sqsubseteq \exists R.B$	$B \sqsubseteq C$	$\exists R.C \sqsubseteq D$	$\models A \sqsubseteq D$
(5)	$A \sqsubseteq \exists S.D$	$S \sqsubseteq R$		$\models A \sqsubseteq \exists R.D$
(6)	$A \sqsubseteq \exists R_1.C$	$C \sqsubseteq \exists R_2.D$	$R_1 \circ R_2 \sqsubseteq R$	$\models A \sqsubseteq \exists R.D$

Reasoning It is a well established result that any $\mathcal{EL}^+$ knowledge base has a least fixed point that can be determined by repeatedly applying a finite set of axioms that produce all entailments of a desired type.(Besold et al. 2017; Kazakov, Krötzsch, and Simančík 2012) In other words, we can say that reasoning in $\mathcal{EL}^+$ often amounts to a sequence of applications of a set of pattern-matching rules. One such set of rules, the set we have used in our experiment, is given in Table 2. The reasoning reaches *completion* when there are no new conclusions to be made. Because people are usually interested most in concept inclusions and restrictions, those are the types of statements we choose to include in our reasoning.

After the reasoning has finished we are able to recursively define supports for each conclusion the reasoner reaches. The first step, of course, only has supports from the knowledge base. After this step supports are determined by effectively running the reasoner in reverse, and replacing each statement that is not in the original knowledge base with a superset that is, as you can see by the colored substitutions in Table 3. When the reasoner proved the last statement it did not consider all of the supports, since it had already proved them. It used the new facts it had learned in the last iteration but we have drawn their supports back out so that we can define a fixed set of inputs from the knowledge base.

Synthetic Data

To provide sufficient training input to our system we provide a synthetic generation procedure that combines a structured forced-lower-bound reasoning sequence with a connected randomized knowledge base. This allows us to rapidly generate many normal semi-random $\mathcal{EL}^+$ knowledge bases of arbitrary reasoning difficulty. For this experiment we choose a moderate difficulty setting so that it can compare with non-synthetic data. An example of one iteration of the two-part sequence is provided in Figure 1. To ensure that the randomized statements do not shortcut this pattern, the random statements are generated in a nearly disjoint space and connected only to the initial seed term. This ensures that at least one element of the random space will also produce random entailments for the duration of the sequence, possibly longer. Our procedure also guarantees that each completion rule will be used at least once every iteration of the sequence so that all reasoning patterns can potentially be learned by the system.

seed_0 = C1 $\sqsubseteq$ C2	C2 $\sqsubseteq$ $\exists$R1.C3
C2 $\sqsubseteq$ C3	C1 $\sqsubseteq$ $\exists$R3.C4
C3 $\sqcap$ C4 $\sqsubseteq$ C5	C2 $\sqsubseteq$ $\exists$R1.C3
C1 $\sqsubseteq$ $\exists$R1.C1	$\exists$R1.C2 $\sqsubseteq$ C4
C1 $\sqsubseteq$ $\exists$R2.C3	R1 $\sqsubseteq$ R2
C2 $\sqsubseteq$ $\exists$R2.C3	R1 $\circ$ R3 $\sqsubseteq$ R4
Which entails	
Step 1:	Step 2:
C1 $\sqsubseteq$ C3	seed_1 = C1 $\sqsubseteq$ C5
C1 $\sqsubseteq$ C4	
C1 $\sqsubseteq$ $\exists$R1.C3	
C1 $\sqsubseteq$ $\exists$R2.C1	
C1 $\sqsubseteq$ $\exists$R4.C4	

Figure 1: First Iteration of Sequence

SNOMED

We have also imported data from the SNOMED 2012 ontology to ensure that our method is applicable to non-synthetic data. SNOMED is a widely-used, publicly available, ontol-

Table 3: Support Generation

	New Fact	Rule	Support
Step 1	C1 ⊑ C3	(1)	C1 ⊑ C2,C2 ⊑ C3
	C1 ⊑ C4	(4)	C1 ⊑ C2,C1 ⊑ ∃R1.C1,∃R1.C2 ⊑ C4
	C1 ⊑ ∃R1.C3	(3)	C1 ⊑ C2,C2 ⊑ ∃R1.C3
	C1 ⊑ ∃R2.C1	(5)	C1 ⊑ ∃R1.C1,R1 ⊑ R2
	C1 ⊑ ∃R4.C4	(6)	C1 ⊑ ∃R1.C1,R1 ∘ R3 ⊑ R4,C1 ⊑ ∃R3.C4
Step 2	C1 ⊑ C5	(2)	C3 ⊓ C4 ⊑ C5,C1 ⊑ C2,C2 ⊑ C3,C1 ⊑ C2,C1 ⊑ ∃R1.C1,∃R1.C2 ⊑ C4

ogy of medical terms and relationships.(De Silva et al. 2011) SNOMED 2012 has 39392 logical axioms, some of which are complex, but this can be normalized in constant time to a logically equivalent set of 124,428 axioms. It is expressed in $\mathcal{EL}^{++}$, and newer versions utilize this complexity more liberally. But the 2012 version contains exactly one statement that is not in $\mathcal{EL}^{+}$ after normalization. We feel confident that omitting $\top \sqsubseteq \exists$topPartOf.Self does not constitute a substantial loss in the ontology.

Because SNOMED is so large, we sample it to obtain connected subsets that also produce a minimum number of reasoning steps. We require that the samples be connected because any normal knowledge base is connected, and it improves the chances that the statements will have entailments. Often, however, random connected samples do not entail anything, and this is also unusual in a normal ontology, so we require a minimum amount of reasoning activity in the samples as well. The reasoning task for SNOMED is more unbalanced than for the synthetic data. It is equally trivial for the reasoner to solve either knowledge base type. However, we observe that random connected sampling tends to favor application of rules 3, 5, and 6 much more heavily than others so the system will have a more difficult time learning the overall reasoning patterns. This imbalance is likely an artifact from SNOMED because it seems to recur in different sample sizes with different settings, though we acknowledge that it could be correlated somehow with the sample generation procedure.

LSTM

LSTMs are type of recurrent neural network that work well with data that has long-term dependencies.(Hochreiter and Schmidhuber 1997) We choose to use an LSTM to learn the reasoning patterns of $\mathcal{EL}^{+}$ because LSTMs are designed to learn sequence patterns. The network is kept as simple as possible to achieve the outputs we require and maximize transparency. For the piecewise system, depicted in Figure 2, we use two separate flat single LSTMs that have the number of neurons matching first the support shape, and later the output shape. The deep system shown in Figure 3 is constructed to match this shape so that the two will be comparable, it simply stacks the LSTMs together so that it can train without supports. We also train a flat system, shown in Figure 4, that has the same input and output shape but lacks the internal supports. We have done this three ways because we can compare the behavior of the systems with support

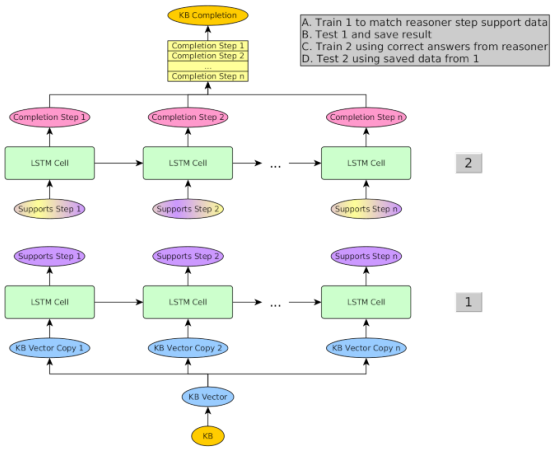


Figure 2: Piecewise Architecture

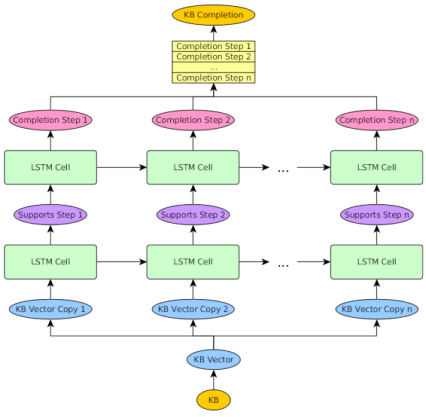


Figure 3: Deep Architecture

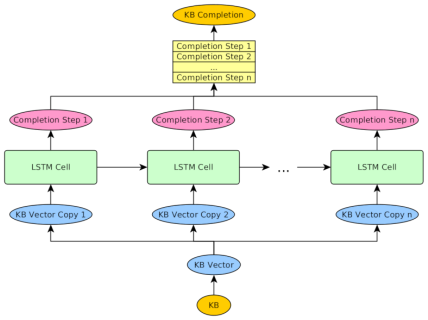


Figure 4: Flat Architecture

against the flat system as a baseline, and then see whether the piecewise training is helping. All designs treat the reasoning steps as the sequence to learn, so the number of LSTM cells is defined by the maximum reasoning sequence length. We train our LSTMs using regression on mean squared error to minimize the distances between predicted answers and correct answers.

Overall System

The individual parts of our system are not themselves novel. Our result, however, lies rather in the unexpected success of this unconventional general approach we have applied. One of the crucial factors in making this work is the way in which we have translated the data from its logical format into something a neural network can understand. In the following sections we will discuss how our system transforms data from a knowledge base format to an LSTM format and back again.

Statement Encoding Every data set we use has a fixed number of names that it can contain for both roles and concepts, and every concept or role has an arbitrary number name identifier. For the synthetic data these numbers are randomly generated. For SNOMED data, all of the labels are stripped and the concept and role numbers are shuffled around and substituted with random integers to remove any possibility of the names injecting a learning bias. These labels are remembered for later retrieval for output but the reasoner and LSTM do not see them. It can seem occasionally like there is a bias in the naming if output for SNOMED data is inspected because all the labels seem related. This is merely a result of our sampling technique which requires connected statements, so it is not a concern.

Knowledge bases have a maximum number of role and concept names that are used to scale all of the integer values for names into a range of $[-1, 1]$. To enforce disjointness of concepts and roles, we map all concepts to $(0, 1]$, all roles to $[-1, 0)$. Each of the six possible normalized axiom forms is encoded into a 4-tuple based on the logical encodings defined in Table 4.

We would like to emphasize the distinction between our method, which we prefer to call an encoding, and a traditional embedding. This encoding adds no additional information to the names beyond the transformation from integer

Table 4: Translation Rules

KB statement		Vectorization
CX $\sqsubseteq$ CY	$\rightarrow$	$[\,0.0,\ \frac{X}{c}, \frac{Y}{c},\ 0.0\,]$
CX $\sqcap$ CY $\sqsubseteq$ CZ	$\rightarrow$	$[\ \frac{X}{c}, \frac{Y}{c}, \frac{Z}{c},\ 0.0\,]$
CX $\sqsubseteq \exists$RY.CZ	$\rightarrow$	$[0.0,\ \frac{X}{c}, \frac{-Y}{r}, \frac{Z}{c}]$
$\exists$RX.CY $\sqsubseteq$ CZ	$\rightarrow$	$[\frac{-X}{r}, \frac{Y}{c}, \frac{Z}{c},\ 0.0]$
RX $\sqsubseteq$ RY	$\rightarrow$	$[0.0, \frac{-X}{r}, \frac{-Y}{r}, 0.0]$
RX $\circ$ RY $\sqsubseteq$ RZ	$\rightarrow$	$[\frac{-X}{r}, \frac{-Y}{r}, \frac{-Z}{r}, 0.0]$

c = Number of Possible Concept Names
r = Number of Possible Role Names

to scaled floating point number. It can be reversed, with a slight loss of precision due to integer conversion, quite directly. The semantics of each encoded predicate name with regards to its respective knowledge base is structural in relation to its position in the 4-tuple, just like it would be in an $\mathcal{EL}^+$ axiom. And its semantics is not at all related to the other non-equal predicate names within the same 4-tuple.

Knowledge Base Encoding In order to input knowledge bases into the LSTM we first apply the encoding defined in the previous section to each of its statements. Then we concatenate each of these encodings end-to-end to obtain a much longer vector. For instance,

$$\text{C1} \sqsubseteq \exists \text{R1.C2},\ \text{R1} \sqsubseteq \text{R2}$$

might translate to

$$[0.0, 0.5, -0.5, 1.0, 0.0, -0.5, -1.0, 0.0].$$

This knowledge base vector is then copied the same number of times as there are reasoning steps and stuck together once again along a new dimension. For an experiment this is done hundreds or thousands of times, and these fill up the tensor that is given to the LSTM to learn. Because some of the randomness can cause ragged data, any tensor dimensions that are not the same length as the maximum are padded with zeros.

Results

Viewed in terms of the questions posed at the beginning of this paper we feel that our results are solid, and they validate a reexamination of how we should approach integrating logic and neural networks. We have attempted to create a system that emulates a reasoning task rather than reasoning output.

If we examine the example output from the synthetic data inputs in Table 5, it is clear that it is getting very close to many correct answers. When it misses, it still appears to be learning the shape, and this makes us optimistic about its future potential. The SNOMED predictions are much more dense and do not fit well into a table, but we have included a few good examples with the original data labels

Table 5: Example Synthetic Output

	Correct Answer	Predicted Answer
Step 0	C9 ⊑ C11	C8 ⊑ C9
	C2 ⊑ C10	C1 ⊑ C9
	C9 ⊑ C12	C8 ⊑ C9
	C7 ⊑ C6	C8 ⊑ ∃R4.C9
	C9 ⊑ ∃R4.C11	C1 ⊑ ∃R5.C9
	C2 ⊑ ∃R4.C9	C8 ⊑ ∃R4.C9
	C9 ⊑ ∃R5.C9	C9 ⊑ ∃R5.C9
	C2 ⊑ ∃R5.C11	
	C9 ⊑ ∃R7.C12	
	C2 ⊑ ∃R6.C12	
Step 1	C9 ⊑ C13	C8 ⊑ C12
	C2 ⊑ C11	C2 ⊑ C10
	C2 ⊑ C12	C1 ⊑ C11
	C2 ⊑ ∃R4.C11	C1 ⊑ ∃R3.C12
	C2 ⊑ ∃R5.C9	C1 ⊑ ∃R4.C8
	C2 ⊑ ∃R7.C12	
Step 2	C2 ⊑ C13	C1 ⊑ C12

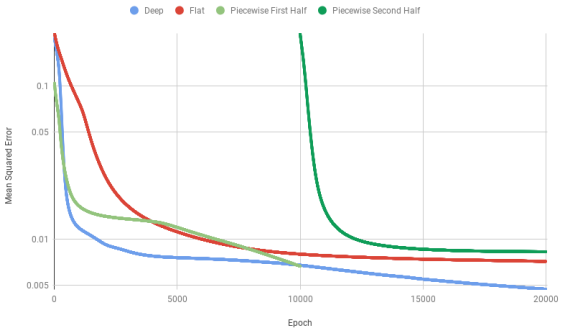


Figure 5: Synthetic Training

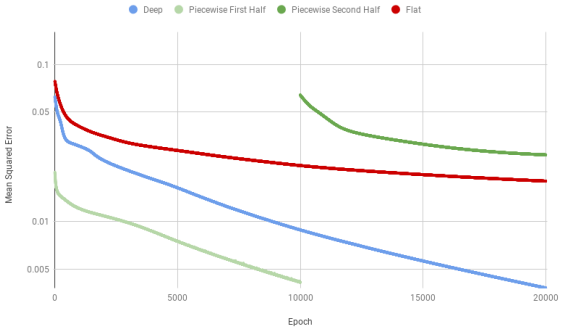


Figure 6: SNOMED Training

Table 6: Example SNOMED Outputs

Correct Answer	C6 ⊑ ∃R4.C1
Meaning	if something is a zone of superficial fascia, then there is a subcutaneous tissue that it is PartOf
Prediction	C6 ⊑ ∃R4.C3
Meaning	if something is a zone of superficial fascia, then there is a subcutaneous tissue of palmar area of little finger that it is PartOf
Correct Answer	C8 ⊑ ∃R3.C2
Meaning	if something is a infrapubic region of pelvis, then there is a perineum that it is PartOf
Prediction	C9 ⊑ ∃R3.C2
Meaning	if something is a zone of integument, then there is a perineum that it is PartOf

translated into English sentences. Because there are so many near-misses we include edit distance evaluations that better capture the degree to which each predicted statement misses what it should have been.

It is interesting to note that by comparing Table 7 with Table 8 we can see that on the much harder SNOMED data the deep system *appears* to have a better result because of the higher F1 score, but the average edit distance, which is our preferred alternative measure for evaluation, is not obviously correlated with the F1-score. This confirms that our fifth research question was appropriate and that F1-score might not be sufficient to evaluate whether or not we can learn the shape of reasoning patterns.

A cause for the discrepancy may be the higher training difficulty for reaching the completion versus reaching the supports in the SNOMED data, which you can see in Figures 5 and 6. We can speculate on the degree to which various factors are contributing to this, but importantly, the fact that this discussion is even possible in the first place is a confirmation that we can answer the fourth research question in the affirmative.

Another interesting result can be seen by examining the charts in Figures 7 - 12. As we have noticed before, the Deep architecture performs quite well on the synthetic data and is quite resistant to increased levels of noise. The flat sys-

Table 7: Average Precision Recall and F1-score For each Distance Evaluation

	Atomic Levenshtein Distance			Character Levenshtein Distance			Predicate Distance		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
	Synthetic Data								
Piecewise Prediction	0.138663	0.142208	0.140412	0.138663	0.142208	0.140412	0.138646	0.141923	0.140264
Deep Prediction	**0.154398**	**0.156056**	**0.155222**	**0.154398**	**0.156056**	**0.155222**	**0.154258**	**0.155736**	**0.154993**
Flat Prediction	0.140410	0.142976	0.141681	0.140410	0.142976	0.141681	0.140375	0.142687	0.141521
Random Prediction	0.010951	0.0200518	0.014166	0.006833	0.012401	0.008811	0.004352	0.007908	0.007908
	SNOMED Data								
Piecewise Prediction	0.010530	0.013554	0.011845	0.010530	0.013554	0.011845	0.010521	0.013554	0.011839
Deep Prediction	**0.015983**	0.0172811	**0.016595**	**0.015983**	0.017281	**0.016595**	**0.015614**	0.017281	**0.016396**
Flat Prediction	0.014414	**0.018300**	0.016112	0.0144140	**0.018300**	0.016112	0.013495	**0.018300**	0.015525
Random Prediction	0.002807	0.006803	0.003975	0.001433	0.003444	0.002023	0.001769	0.004281	0.002504

Table 8: Average Statement Edit Distances with Reasoner

	Atomic Levenshtein Distance			Character Levenshtein Distance			Predicate Distance		
	From	To	Average	From	To	Average	From	To	Average
	Synthetic Data								
Piecewise Prediction	1.336599	1.687640	1.512119	1.533115	1.812006	1.672560	2.633427	4.587382	3.610404
Deep Prediction	**1.256940**	**1.507150**	**1.382045**	**1.454787**	**1.559751**	**1.507269**	**2.504496**	**3.552074**	**3.028285**
Flat Prediction	1.344946	1.584674	1.464810	1.586281	1.660409	1.623345	2.517655	3.739770	3.128713
Random Prediction	1.598016	1.906369	1.752192	1.970604	1.289533	1.630068	5.467918	10.57324	8.020583
	SNOMED Data								
Piecewise Prediction	1.704931	**2.686562**	**2.195746**	2.016249	2.862737	2.439493	6.556592	**5.857769**	6.207181
Deep Prediction	1.759633	3.052080	2.405857	2.027190	3.328850	2.678020	**4.577427**	6.179389	**5.378408**
Flat Prediction	**1.691738**	2.769542	2.230640	**1.948757**	2.991328	2.470042	5.548226	6.665659	6.106942
Random Prediction	1.814656	3.599629	2.707143	2.094682	**1.621700**	**1.858191**	5.169093	12.392325	8.780709

tem gets similar results, but slightly worse. Surprisingly, the piecewise model starts out mostly the same as the deep system on synthetic data but then very quickly falters. When we examine the results for the harder SNOMED data, however, the opposite seems to be happening. The deep and flat systems still track each other. But for the unbalanced data the deep system crosses into the random range by 30-50% corruption, while the piecewise system is able to skirt just above random for the Character and Predicate distance functions until around 80-90%.

Methodology

Our system is trained using randomized 10-fold cross validation to 20000 epochs on the deep and flat systems and 10000 epochs each for the parts of the piecewise system at a learning rate of 0.0001. The results from every fold are saved and we report the average of all of the runs. Often there are better and worse runs in any given 10-fold validation but the averages across multiple cross-validations with the same settings remain basically the same. 10-fold cross-validations are performed with an extra comparison against data that has been corrupted at a fixed rate, starting with zero percent probability of corruption and moving upwards by ten percent each time. The data in Tables 7 and 8 is from the comparison with no corruption, so the error data is not reported (it is identical with reasoner answers).

To perform our evaluations we use three unique distance measurements. We have a naive “Character” Levenshtein distance function that takes two unaltered knowledge base statement strings and computes their edit distance.(Wikibooks contributors 2019 accessed November 19 2019) However, because some names in the namespace are one digit numbers and other names are two digit numbers, we include a modified version of this function, called ”Atomic”, that uniformly substitutes all two digit numbers in the strings with symbols that do not occur in either. Since there cannot be more than eight unique numbers in two decoded strings there are no issues with finding enough new symbols. By doing the substitutions we can see the impact that the number digits were having on the edits from the Atomic Levenshtein distance. Finally we devise a distance function that is based on our encoding scheme. The Predicate Distance method disassembles each string into only its predicates. Then, for each position in the 4-tuple, a distance is calculated that yields zero for perfect matches, absolute value of guessed number - actual number for correct Class and Role guesses, and guessed number + actual number for incorrect Class and Role matches. So, for instance, guessing C1 when the answer is C2 will yield a Predicate Distance of 1, while a guess of R2 for a correct answer of C15 will yield 17. Though this method is specific to our unique encoding, we believe it detects good and bad results quite well because perfect hits are 0, close misses are penalized a little, and large misses are penalized a lot.

For each method we take every statement in a knowledge base completion and compare it with the best match in the reasoner answer, random answers, and corrupted knowledge base answers. While we compute these distances we are able

to obtain precision, recall, and F1-score by counting the the number of times the distance returns zero and treating the statement predictions as classifications. Each time the system runs it can make any number of predictions, from zero to the maximum size of the output tensor. This means that, although the predictions and reasoner are usually around the same size and those comparisons are fine, we have to generate random data to compare against that is as big as could conceivably be needed by the system. Any artificial shaping decisions we made to compensate for the variations between runs would invariably introduce their own bias in how we selected them. Thus the need to use the biggest possible random data to compare against means the precision, recall, and F1-score for random are low.

Conclusion

In this experiment demonstrate that $\mathcal{EL}^+$ reasoning can be emulated with an LSTM. Although our example is merely a prototype, we can definitively say yes to the first research question, a neural network can be trained to emulate completion reasoning behavior. And because we have answered this first research question without using name adjustments or embeddings, so we can also answer yes to the second and third questions. We have already discussed questions four and five.

Future Work

There are a number of potential improvements that could build on the work presented here. A few of the choices we made might have had an impact of the results of the study and we are curious if alternate strategies could improve the way that the system learns reasoning patterns. We might have chosen a 3-tuple encoding pattern that was more dense but did not mirror the shape of the statements symmetrically. We also considered adding the statements as an additional dimension rather than concatenating them all together. This may have made it easier for the system to distinguish individual statements but would also add a lot of complexity in the extra dimension. We experimented with different neuron types like gated recurrent unit (GRU) and basic RNN, though these had no significant impact on the results. The type of neuron seems to have less effect than the overall structure of the network. It would be interesting to see if different ways of segmenting a network, besides just along the supports as we have defined them, could improve results.

Along with these ideas to improve our current system, we also speculate that with some effort we should be able to accomplish basically the same type of thing with a more expressive logic and a new encoding. $\mathcal{EL}^{++}$ for instance would be an easy first attempt since it is so similar to $\mathcal{EL}^+$. Though we should be able to adapt this strategy for any logic that uses completion rules for reasoning. Whatever we choose to do with it, we are optimistic that we can use this strategy to advance understanding of how logic and neural networks can work together.

Testing Environment

All testing was done on a computer running Ubuntu 19.10 64-bit with an Intel Core i7-9700K CPU@3.60GHz x 8, 47.1 GiB DDR4, and a GeForce GTX 1060 6GB/PCIe/SSE2. Source code and experiment data is available on GitHub https://github.com/aaronEberhart/PySynGenReas.

Acknowledgements This work was partially supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386, and by the National Science Foundation under award number 1936677.

References

Besold, T. R.; d'Avila Garcez, A. S.; Bader, S.; Bowman, H.; Domingos, P. M.; Hitzler, P.; Kühnberger, K.; Lamb, L. C.; Lowd, D.; Lima, P. M. V.; de Penning, L.; Pinkas, G.; Poon, H.; and Zaverucha, G. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR* abs/1711.03902.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, 2787–2795.

Das, R.; Neelakantan, A.; Belanger, D.; and McCallum, A. 2016. Chains of reasoning over entities, relations, and text using recurrent neural networks. *CoRR* abs/1607.01426.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; and McCallum, A. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*.

De Silva, T. S.; MacDonald, D.; Paterson, G.; Sikdar, K. C.; and Cochrane, B. 2011. Systematized nomenclature of medicine clinical terms (snomed ct) to represent computed tomography procedures. *Comput. Methods Prog. Biomed.* 101(3):324–329.

Ebrahimi, M.; Sarker, M. K.; Bianchi, F.; Xie, N.; Doran, D.; and Hitzler, P. 2018. Reasoning over rdf knowledge bases using deep learning. *arXiv preprint arXiv:1811.04132*.

Hochreiter, S., and Schmidhuber, J. 1997. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, 473–479.

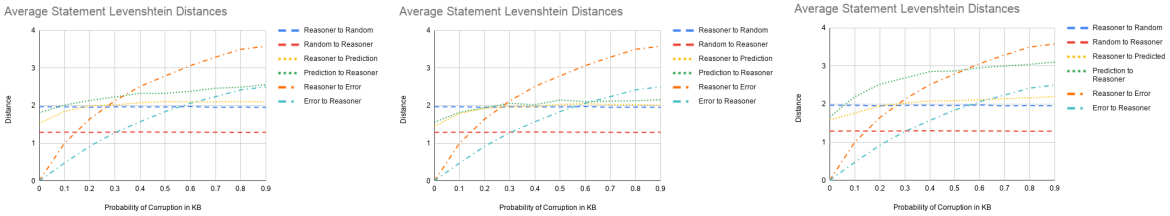
Kazakov, Y.; Krötzsch, M.; and Simančík, F. 2012. Elk: a reasoner for owl el ontologies. *System Description*.

Kulmanov, M.; Liu-Wei, W.; Yan, Y.; and Hoehndorf, R. 2019. El embeddings: Geometric construction of models for the description logic el++. *arXiv preprint arXiv:1902.10499*.

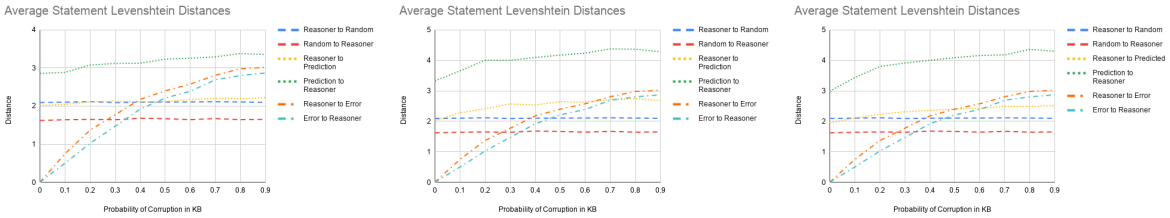
Makni, B., and Hendler, J. 2018. *Deep Learning for Noise-tolerant RDFS Reasoning*. Ph.D. Dissertation, Rensselaer Polytechnic Institute.

Wikibooks contributors. 2019 (accessed November 19, 2019). Algorithm implementation/strings/levenshtein distance.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*.

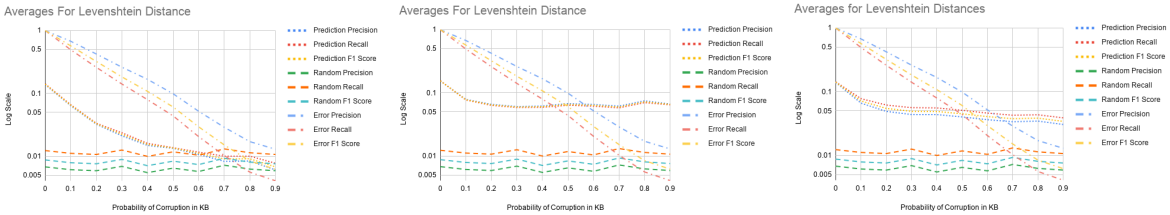


(a) Synthetic Data Piecewise Architecture (b) Synthetic Data Deep Architecture (c) Synthetic Data Flat Architecture

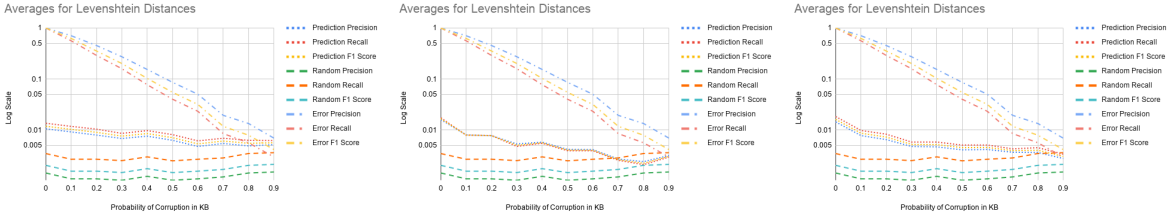


(d) SNOMED Data Piecewise Architecture (e) SNOMED Data Deep Architecture (f) SNOMED Data Flat Architecture

Figure 7: Character Levenshtein Distance

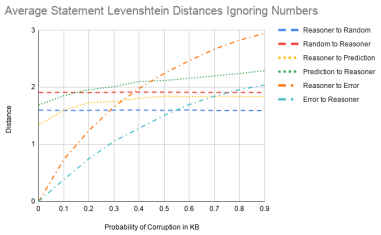


(a) Synthetic Data Piecewise Architecture (b) Synthetic Data Deep Architecture (c) Synthetic Data Flat Architecture

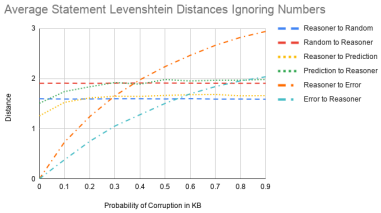


(d) SNOMED Data Piecewise Architecture (e) SNOMED Data Deep Architecture (f) SNOMED Data Flat Architecture

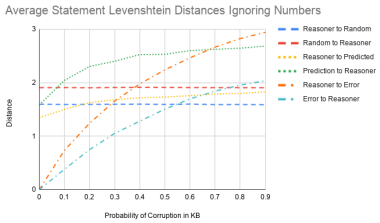
Figure 8: Character Levenshtein Distance Precision, Recall, and F1-score



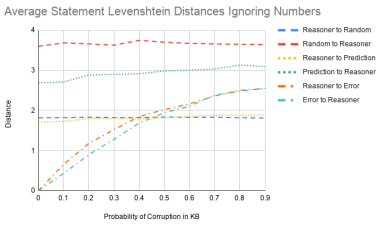
(a) Synthetic Data Piecewise Architecture



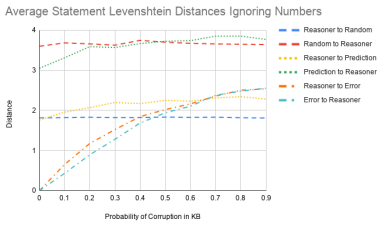
(b) Synthetic Data Deep Architecture



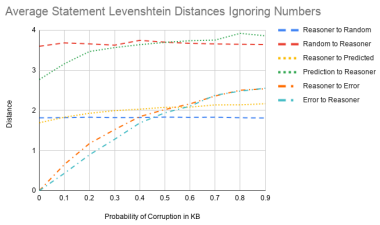
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

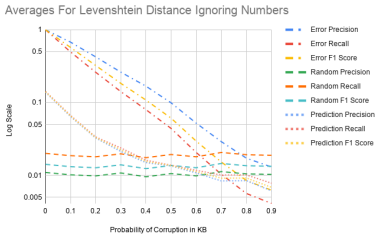


(e) SNOMED Data Deep Architecture

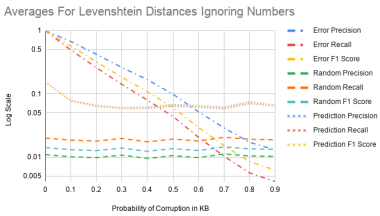


(f) SNOMED Data Flat Architecture

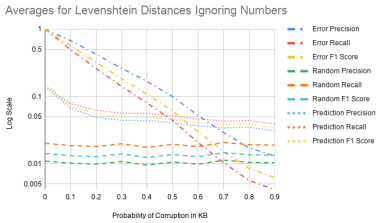
Figure 9: Atomic Levenshtein Distance



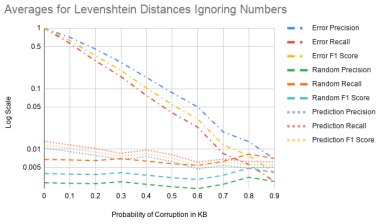
(a) Synthetic Data Piecewise Architecture



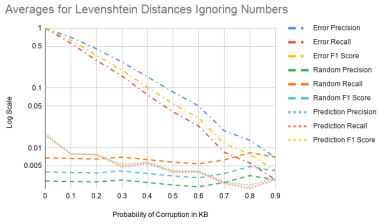
(b) Synthetic Data Deep Architecture



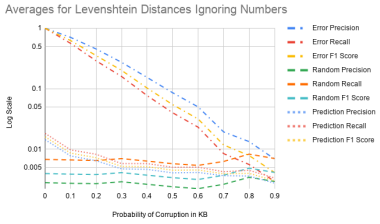
(c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture

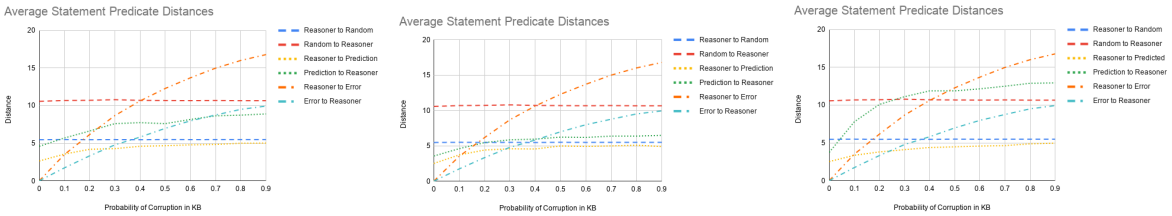


(e) SNOMED Data Deep Architecture

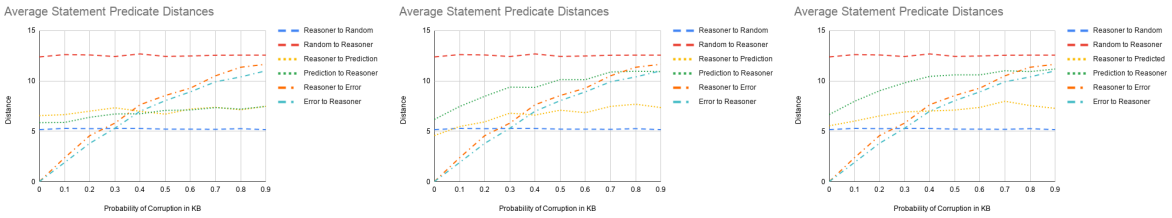


(f) SNOMED Data Flat Architecture

Figure 10: Atomic Levenshtein Distance Precision, Recall, and F1-score

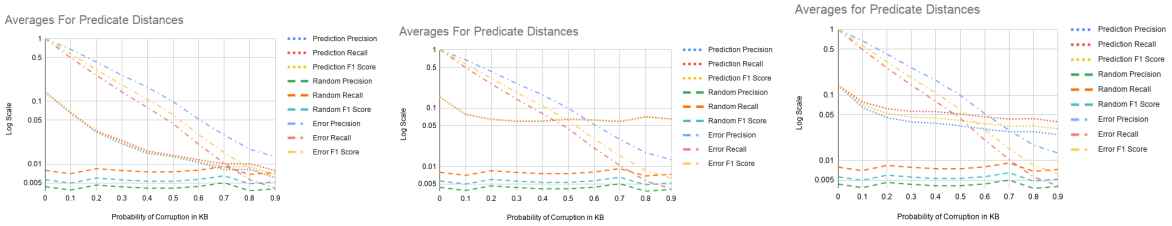


(a) Synthetic Data Piecewise Architecture (b) Synthetic Data Deep Architecture (c) Synthetic Data Flat Architecture

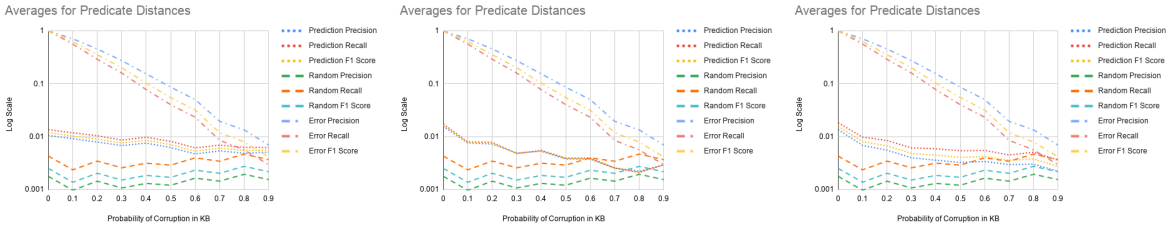


(d) SNOMED Data Piecewise Architecture (e) SNOMED Data Deep Architecture (f) SNOMED Data Flat Architecture

Figure 11: Predicate Distance



(a) Synthetic Data Piecewise Architecture (b) Synthetic Data Deep Architecture (c) Synthetic Data Flat Architecture



(d) SNOMED Data Piecewise Architecture (e) SNOMED Data Deep Architecture (f) SNOMED Data Flat Architecture

Figure 12: Predicate Distance Precision, Recall, and F1-score

Expressibility of OWL Axioms with Patterns

Aaron Eberhart✉, Cogan Shimizu, Sulogna Chowdhury, Md. Kamruzzaman Sarker, and Pascal Hitzler

Data Semantics Lab, Kansas State University, USA
aaroneberhart@ksu.edu

Abstract. The high expressivity of the Web Ontology Language (OWL) makes it possible to describe complex relationships between classes, roles, and individuals in an ontology. However, this high expressivity can be an obstacle to correct usage and wide adoption. Past attempts to ameliorate this have included the development of specific, presumably human-friendly syntaxes, such as the Manchester syntax or graphical interfaces for OWL axioms, albeit with limited success. If modelers want to develop suitable OWL axioms it is important to make this as easy as possible.
In this paper, we adopt an idea from the Protégé plug-in, OWLAx, which provides a simple, clickable interface to automatically input axioms of a limited number of types by following simple axiom patterns. In particular, each of these axiom patterns contains at most three classes or roles. We hypothesize that most of the axioms in existing ontologies could be expressed semantically in terms of simple patterns like these, which would mean that more complex patterns can be used very sparingly.
Our findings, based on an analysis of 518 ontologies from six public ontology repositories, confirm this hypothesis: Over 90% of class axioms in the average ontology are indeed expressible with our simple patterns. We provide a detailed analysis of our findings.

1 Introduction

Knowledge graph schema are complex artifacts that can be difficult and expensive to produce and maintain. This is especially true when encoding them in OWL (the Web Ontology Language) as ontologies. The high expressivity of OWL is a boon, in that it makes it possible to describe complex relationships between classes, roles,[1] and individuals in an ontology. At the same time, however, this high expressivity is often an obstacle to its correct usage that can limit adoption. Past attempts to ameliorate this have included the development of specific, presumably human-friendly syntaxes, such as the Manchester syntax [10], or graphical interfaces for OWL axioms, albeit with modest success [16]. Additionally, certain engineering paradigms and methodologies have been developed, such as eXtreme Design [2] or Modular Ontology Modeling [7,19], that try to simplify the modeling process.

[1] We refer to properties as **roles**, unless a distinction is relevant, as this is the standard description logic term. These include both object properties and data properties.

In general, these methodologies aim to guide ontology developers through the complex modeling process by either abstracting the complexity away (for example, through the use of Ontology Design Patterns), or by limiting the scope of the model to something immediately applicable and understandable. In this paper we are particularly interested in the latter, especially during the axiomatization process. We believe that it is important to investigate new avenues for improving the approachability of creating suitable OWL axioms.

One of the core tenets of the Modular Ontology Modeling methodology is to produce schema diagrams and then systematically axiomatize them, with the input of domain experts. This systematic axiomatization is inspired by the OWLAx plugin for Protégé,[2] which provides a simple, clickable interface to automatically input axioms of limited syntactic forms that are all created from simple axiom patterns [17]. In particular, each of these simple axiom patterns contains at most three classes or roles. In [17], it was posited (but not demonstrated) that the 17 axiom patterns provided by the interface were sufficient for *most* modeling purposes. In this paper, we test that hypothesis by analyzing 518 ontologies from six public ontology repositories. Concretely, we show the following:

H1. Almost all axioms in OWL ontologies are expressible using a set of simple axiom patterns, like those found in Table 1.

And indeed, as we will see, it holds for over 90% of class axioms in the average ontology using our relatively straightforward analysis. With a more thorough analysis or with different patterns, the percentage may even be higher.

2 Related Work

We are aware of only a very limited amount of research that specifically concerns the semantic, not syntactic, composition and expressibility of ontologies regarding patterns. There are several studies, such as [5,23,14], which investigate the use of OWL syntax and constructs in general. However, a mere syntactic survey of OWL as it is used in practice does not directly address the question we are investigating, namely whether a relatively small set of axiom patterns suffices to express most OWL axioms. Zhang et al. [26] look at ways to measure the design complexity of ontologies. Their work is focused more on ontology quality evaluation than ontology composition. Some have also attempted to measure the effect that axioms like existential quantifiers have on reasoning time, such as Kang et al. [11], although it is only tangentially related to the work that we are presenting.

There are also, as previously mentioned, tools that attempt to simplify OWL ontology development, such as Manchester Syntax [10], WebVOWL [13], CoModIDE [18], Graffoo [3], and ROWLTab [16]. These tools simplify the development process but they do not measure whether OWL axioms are necessarily complex

[2] See `https://protege.stanford.edu/`.

in everyday usage. It could very well be the case that OWL is unavoidably complicated and these tools are needed to deal with this complexity, although we believe our work demonstrates that this is usually not the case.

3 Methodology

Our hypothesis is that most axioms in ontologies could be expressed with simple axiom patterns. In this section, we will define what we mean by simple axioms, then give an example of a set of patterns that generate simple axioms, such as those used in the Protégé plugin OWLAx. Following that, we will describe how to measure the extent to which an ontology is expressible with simple axiom patterns, and then provide some minimal normalizations for ontologies which we will use in our evaluation.

3.1 Simple Axioms

The simple axioms we study in this paper are defined below. We consider description logic syntax for OWL DL, that is, we identify it with the description logic $\mathcal{SROIQ}(D)$ [8].

Definition 1. *A **Simple Axiom** is any OWL axiom that contains at most three class or role names, or a data range, and is not a syntactic shortcut for other OWL axioms as defined in the OWL 2 Specification.*[3] *Any axiom which is not simple is a **Complex Axiom**.*

Our set of axiom patterns is designed for class axioms, so we restrict our focus to class axioms in the evaluation, although, in principle, the notion of a simple axiom could apply to role (RBox) axioms as well. The limitation of three atoms for simple axioms is an intuitive threshold, in terms of size, because it means that nesting is limited, yet the axiom can still contain expressions and participate in complex inferences in combination with other simple axioms. This would not be the case for axioms limited to size two, where one could only express $A \sqsubseteq B$ for classes, or $R \sqsubseteq S$ for roles, which would radically limit the expressivity of the ontology. Axioms with more than three atomic classes or roles may be more expressive, but are often equivalent through normalization to smaller axioms, so they do not make not good candidates for simple axioms. Note that negation and inverse are not considered complex, since the definition considers only the number of names; of course double negation can be eliminated trivially.

3.2 OWLAx axiom patterns

OWLAx [17] is a Protégé plugin that allows users to automatically generate certain simple OWL axioms using a graphical interface. The set of axioms we study in this paper are inspired by the axioms that OWLAx can create, and they are listed in Table 1.

[3] See `https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`

Subclass	$A \sqsubseteq B$	Functional	$\top \sqsubseteq \leqslant 1R.\top$
Disjoint Classes	$A \sqcap B \sqsubseteq \bot$	Qualified Functional	$\top \sqsubseteq \leqslant 1R.B$
Domain	$\exists R.\top \sqsubseteq B$	Scoped Functional	$A \sqsubseteq \leqslant 1R.\top$
Scoped Domain	$\exists R.A \sqsubseteq B$	Qualified Scoped Functional	$A \sqsubseteq \leqslant 1R.B$
Range	$\top \sqsubseteq \forall R.B$	Inverse Functional	$\top \sqsubseteq \leqslant 1R^{-}.\top$
Scoped Range	$A \sqsubseteq \forall R.B$	Inverse Qualified Functional	$\top \sqsubseteq \leqslant 1R^{-}.B$
Existential	$A \sqsubseteq \exists R.B$	Inverse Scoped Functional	$A \sqsubseteq \leqslant 1R^{-}.\top$
Inverse Existential	$A \sqsubseteq \exists R^{-}.B$	Inverse Qualified Scoped Functional	$A \sqsubseteq \leqslant 1R^{-}.B$
		Structural Tautology	$A \sqsubseteq \geqslant 0R.B$

A,B,R are variable terms that contain at most one class or role name, or a data range

Table 1: OWLAx Axiom Patterns

The actual implementation details of the OWLAx plugin are not pertinent to our discussion. Rather, we are interested in what it happens to contain: a set of patterns that only make simple axioms. In this sense, the axiom patterns are *simple axiom patterns*, since they can generate only simple axioms. And because it was designed specifically to help create ontologies, we speculate that ontologies will be mostly expressible using these patterns. We now discuss how to assess the extent to which an ontology can be expressed using such *axiom patterns*.

3.3 Axiom Pattern Expressibility

To study whether axioms in an ontology are expressible with simple axiom patterns, we first define the term axiom pattern and then show how a set of axiom patterns can be used to study axioms in an ontology, obtaining multiple metrics to evaluate pattern expressibility.

Definition 2. *An **Axiom Pattern** is a programmatic template for creating new, syntactically correct axioms. An axiom pattern may have variable terms that can be used to obtain specific axioms by substitution. Given an axiom α and a pattern p, we say that p* can generate *α (or, p* is *α*-generating*) if α can be obtained by appropriately substituting variable terms in p.*

For example, the axiom pattern $A \sqsubseteq \exists R.B$ for an existential axiom from Table 1, where A, B, R are variable terms, can be used to generate the axiom $\text{Dog} \sqsubseteq \exists \text{chases.Squirrel}$ by substitution, where "Dog" and "Squirrel" are classes and "chases" is a role. Note that axiom patterns are very different from Ontology Design Patterns (ODPs) [20], because an ODP is a partial ontology representing a generic solution to a recurring ontology modeling problem, while an axiom pattern is a pattern for making single axioms. They are fundamentally different in purpose and nature (although the term *pattern* can be used for either).

Definition 3. *The **Axiom Pattern Expressibility** $ae_{\mathcal{P}}(\alpha)$ of an axiom α w.r.t. a set of axiom patterns $\mathcal{P}$ is the set of patterns $p \in \mathcal{P}$ each of which can generate α with the fewest substitutions. Formally, given an axiom α and a*

pattern p that can generate α, let $s_p(\alpha)$ be the number of substitutions required to generate α from p. Given an axiom α, and a set $\mathcal{P}$ of axiom patterns, let $ae_{\mathcal{P}}(\alpha)$ be the set of α-generating patterns from $\mathcal{P}$ such that, for all $q \in ae_{\mathcal{P}}(\alpha)$ and $p \in \mathcal{P}$, we have $s_q(\alpha) \leq s_p(\alpha)$. Note that $ae_{\mathcal{P}}(\alpha)$ may be empty if there are no α-generating patterns in $\mathcal{P}$.

We give an example for these definitions. Let P be the set of axiom patterns from Table 1, and let α be Human $\sqsubseteq \leq 1$.hasHeart.$\top$. Then $s_{A \sqsubseteq \leq 1R.\top}(\alpha) = 2$ and $s_{A \sqsubseteq \leq 1R.B}(\alpha) = 3$. It is easy to check that no other patterns in P can generate α. Hence $ae_{\mathcal{P}}(\alpha) = \{A \sqsubseteq \leq 1R.\top\}$.

Proposition 1. *For $\mathcal{P}$ the set of axiom patterns from Table 1, and given any axiom α in $\mathcal{SROIQ}(D)$, $ae_{\mathcal{P}}(\alpha)$ is either empty or a singleton set. I.e. if an axiom can be generated from a pattern from P, then there is a unique pattern with the minimal number of required substitutions.*

Proof. It is sufficient to show that no axiom can be created by more than one of our patterns with the fewest substitutions. This can be verified easily by inspecting Table 1 and comparing pairs of patterns. We draft the thought process. Table 1 states that each variable term contains at most one class or role name, therefore subclass, disjoint classes, and structural tautology patterns all have no overlapping patterns which could also produce axioms of those forms. The domain, range, and existential patterns do not mutually overlap except in pairs that vary only in the location of $\top$ or $^-$, so the claim is clearly also true, since $\top$ or $^-$ in the pattern reduces the number of substitutions required to generate an axiom containing it by 1. Functional and inverse functional patterns follow a similar structure, where an axiom is always uniquely obtainable with the fewest substitutions from the pattern containing $\top$ and $^-$ in the same locations.

The following will be used in our evaluations.

Definition 4. *The **Average Axiom Pattern Expressibility** $\overline{ae}_{\mathcal{P}}(\mathcal{A})$ for a set of axioms $\mathcal{A}$ of cardinality $|\mathcal{A}|$ (i.e., $|\mathcal{A}|$ is the number of axioms in $\mathcal{A}$) is defined as*

$$\overline{ae}_{\mathcal{P}}(\mathcal{A}) = \frac{1}{|\mathcal{A}|} \sum_{\alpha \in \mathcal{A}} |ae_{\mathcal{P}}(\alpha)|.$$

*The **Average Ontology Axiom Pattern Expressibility** $\overline{oe}_{\mathcal{P}}(\mathcal{O})$ of a set of ontologies $\mathcal{O}$ having cardinality $|\mathcal{O}|$ and set of axiom patterns $\mathcal{P}$, is given by*

$$\overline{oe}_{\mathcal{P}}(\mathcal{O}) = \frac{1}{|\mathcal{O}|} \sum_{\mathcal{A} \in \mathcal{O}} \overline{ae}_{\mathcal{P}}(\mathcal{A}),$$

where $\mathcal{A}$ represents the set of axioms in each ontology.

It is important to note that average ontology axiom pattern expressibility can be used to evaluate a set of ontologies, and average axiom pattern expressibility can be used to evaluate any number of axioms, e.g. a set of axioms which has been collected from multiple ontologies.

3.4 Normalization

We have discussed our evaluation measures for axiom pattern expressibility of an ontology. However, there remains an issue that ontologies often vary radically in the way they are syntactically expressed, even if semantically they mean similar or even equivalent things. Ontologies are written for completely different purposes and at differing levels of complexity; some ontologies are developed for complex reasoning applications, while others are used for more straightforward data integration. Even within a single ontology, different authors may express equivalent statements in different ways based on personal preference or style. To give an example, class disjointness of two classes A and B can be expressed with any of the (equivalent) axioms $A \sqcap B \sqsubseteq \bot$, $A \sqsubseteq \neg B$, $B \sqsubseteq \neg A$, and others.

Hence, in order to evaluate the pattern expressibility of a large number of ontologies uniformly, we therefore need at least a minimal syntactic normalization strategy taken from community standards that allows us to compare disparate sources without biasing the evaluation in favor of any particular style. For this, we use multiple strategies derived from common OWL practices.

Our normalization begins by filtering out all axioms except class, role, and HasKey axioms. This is necessary because there are many OWL axioms for which our pattern study will not apply. Included in this are assertion (ABox) axioms, since these are primarily axioms about instances rather than classes and roles, but also axioms such as annotations, declarations, and datatype definitions, that are axioms according to the OWL 2 specification but carry no or few formal semantics. HasKey axioms are taken into account because they are logical axioms and not assertions or annotations, although their semantics is different from class and role axioms. Note that none of our axiom patterns matches HasKey axioms. The remaining class and role axioms are then transformed according to the following procedures.

The first transformation that we perform is an equivalence transformation based on the syntactic shortcuts defined in the OWL Structural Specification [15]. Whenever an axiom is found that has one of the forms in Column 1 of Table 2, we perform the designated substitution. These substitutions are equivalent rewritings so they do not alter the semantics of the ontology. It is also possible that other simple transformations of class axioms according to the equivalences defined in the structural specification could improve the evaluation, since these axioms also might be expressible using patterns. Our transformation thus may lead to an undercount in our disfavor; we will come back to this point later. Thus, for EquivalentClasses, DisjointClasses, and DisjointUnion we convert them to sets of SubClass axioms using definitions in the OWL 2 specification.

The second transformation that we perform is obtaining negation normal form (NNF) of all class axioms in an ontology. By using the NNF we can transform all of the class axioms in an ontology into simple syntactic forms that are stripped of unnecessary information that might be due to coincidence rather than semantic equivalence.

The last transformation we apply is splitting SubClass axioms with conjunctions in the consequent, or disjunctions in the antecedent, into separate axioms.

Ontology Axiom	Substituted Axiom
ReflexiveObjectProperty(R)	$\top \sqsubseteq \exists R.\mathbf{Self}$
IrreflexiveObjectProperty(R)	$\exists R.\mathbf{Self} \sqsubseteq \bot$
FunctionalObjectProperty(R)	$\top \sqsubseteq \leqslant 1R.\top$
FunctionalDataProperty(S)	$\top \sqsubseteq \leqslant 1S.\top$
InverseFunctionalObjectProperty(R)	$\top \sqsubseteq \leqslant 1R^{-}.\top$
ObjectPropertyRange(R C)	$\top \sqsubseteq \forall R.C$
DataPropertyRange(S D)	$\top \sqsubseteq \forall S.D$
ObjectPropertyDomain(R C)	$\exists R.\top \sqsubseteq C$
DataPropertyDomain(S C)	$\exists S.\top \sqsubseteq C$

R is an ObjectProperty, S is a DataProperty, C is a Class, and D is a DataRange

Table 2: Axiom Transformations

This is a standard procedure in many normalizations, and we simply replace the axiom with a set of axioms formed from the conjuncts or disjuncts whenever an axiom of this type is found. There is a special case that occurs only when the consequent is an ExactCardinality expression whose value is equal to 1. In this case, we do not use a MinCardinality 1 substitution but instead add an existential, since that is equivalent and more compact.

All normalizations are performed sequentially and axioms are output into a separate collection for evaluation. This compartmentalizes the data and ensures that no duplicates are created, even when a single axiom is transformed into a set of axioms. The sets of axioms $\mathcal{A}$ used in our evaluations are these separate normalized sets.

4 Evaluation

We analyze a set of 518 ontologies from various sources, normalizing them, and testing them for axiom pattern expressibility according to the principles described in the previous section. Ontologies were selected from diverse sources with unique design requirements: benchmark ontologies, Ontology Design Patterns (ODPs), ontologies extracted from Linked Open Vocabulary (LOV) [22], as well as medical domain ontologies. It would technically be possible to integrate other types of linked data in this analysis, though semantically it may not be straightforward to interpret the results if the data was mixed, so we use only OWL ontologies. Statistics about the original ontologies gathered before and after normalization can be found in Table 3. The normalization adds a few axioms to the set of axioms from an ontology whenever an axiom is split, so counts increase by a factor of around 7-8 to 10, except for ontologies that contain many assertions that were excluded during the normalization process, like hydrography and anatomy benchmarks. As mentioned previously, the normalized axioms are the axioms used in the evaluation and the equations. We use the set of all normalized axioms from all ontologies for the $\overline{ae}_{\mathcal{P}}(\mathcal{A})$ numbers and the set of normalized axioms from each ontology for $\overline{oe}_{\mathcal{P}}(\mathcal{O})$ numbers.

	LOV	Hydrography	Anatomy	Conference	ODP	Ontobee	Misc
classes	20075	341	6048	498	577	509925	80796
roles	10106	121	5	226	600	10453	535
data properties	8340	77	0	85	71	480	66
axioms	274991	8873	41407	3037	7378	5071892	816572
logical axioms	82450	5463	16383	2153	3223	963880	272334
normalized axioms	94937	1748	9951	2774	3917	1208950	391015
ontologies	250	4	2	7	80	171	4

Table 3: Ontology Statistics. Logical axioms are axioms that are neither declarations nor annotations

In this section, we report the result for all ontologies we tested, then go into details about each source, reporting a separate evaluation for each. Next we break down the results by profile and report the numbers for those as well. In all cases, the expressibility numbers are reported for All Axioms, Class Axioms, and Simple Class Axioms. Our axiom patterns can only express simple class axioms, thus the values for 'All Axioms' represent the evaluation using all class, role, and HasKey axioms, 'Class Axioms' indicates the evaluation using only class axioms, and 'Simple Class Axioms' represents the evaluation for only simple class axioms. In Figures and Tables, the term "miss" is used to indicate a simple axiom that was observed but was inexpressible using our patterns. The last value we report, which is a byproduct of calculations that produce expressibility numbers, is the percent subclass and percent existential, as well as their combination. By this we mean, what percent of all of the axioms in an ontology are expressible with the subclass pattern, the existential pattern, or both. It will turn out in nearly every case that a surprisingly high proportion of most ontologies is expressible with just these two simple axiom patterns. OWL files and source code for the evaluation, except the gene ontology which can't be uploaded due to size restrictions, can be found on the GitHub page `https://github.com/aaronEberhart/owlax` and the raw data can be inspected in the spreadsheet at `https://tinyurl.com/eswc2021`.

4.1 Overall Expressibility

The average axiom pattern expressibility and the average ontology axiom pattern expressibility for our simple axiom patterns over all normalized axioms in all ontologies is included in Table 4, as well as the standard deviation for the ontology axiom pattern expressibility.

Figure 1 shows the overall distribution of axioms for the entire collection of ontologies. Complex class axioms, role axioms, miss (inexpressible simple axioms) and HasKey are the axioms that cannot be generated by our patterns; note that only the first two of these play a significant role. Simple subclass is 54.8%, and existential is 23.9%, totaling 78.8%. This is almost the same as the axiom expressibility value for all axioms (82.2%). A more detailed view of axiom type distributions can be found in the next section in Figure 2.

	$\overline{ae}_{\mathcal{P}}(\mathcal{A})$	$\overline{oe}_{\mathcal{P}}(\mathcal{O})$	StdDev $\overline{oe}_{\mathcal{P}}(\mathcal{O})$
All Axioms	82.2%	82.9%	0.206
Class Axioms	83.8%	92.9%	0.165
Simple Class Axioms	99.8%	96.5%	0.153

Table 4: Overall Average Expressibility

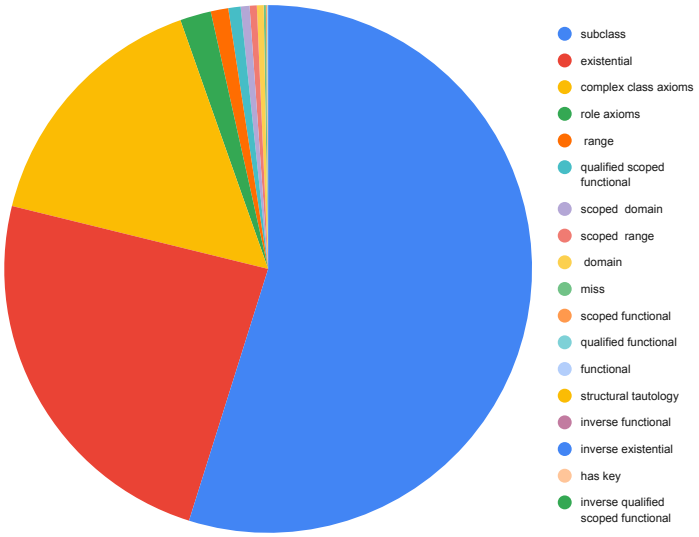


Fig. 1: Overall Distribution of Axioms

4.2 Source Expressibility

As they are all from very different domains, each source was analyzed independently from the whole. Our evaluation includes 250 ontologies that were automatically pulled from LOV using a script that can be found on the project GitHub. We obtained benchmark ontologies that are used for ontology alignment evaluation. There are 4 ontologies from Hydrography, 2 from Anatomy, and 7 from the Conference domains, and each appear in their own column in Tables 5 and 6. We also obtained and evaluated 80 ODPs, as well as a collection of 171 OWL files that are mainly from the medical domain from the ontobee[4] [25] website. Additionally, we gathered 4 ontologies that did not fall neatly into any of these categories but nonetheless seemed interesting to include in the overall result. These ontologies are General Formal Ontology [6], Gene Ontology [4], GeoLink Base Ontology [12], and the Enslaved Ontology [21], and their average is labeled Misc in the tables.

[4] `http://ontobee.org`.

	LOV	Hydrography	Anatomy	Conference	ODP	Ontobee	Misc
All Axioms	78.7%	77.1%	99.9%	89.3%	85.7%	88.7%	62.4%
Class Axioms	92.6%	84.5%	100%	96.2%	97.3%	89.9%	62.5%
Simple Class Axioms	99.2%	96.5%	100%	99.2%	99.1%	99.8%	99.9%

Table 5: $\overline{ae}_{\mathcal{P}}(\mathcal{A})$ By Source

	LOV	Hydrography	Anatomy	Conference	ODP	Ontobee	Misc
All Axioms	81.3%	77.5%	99.9%	87.3%	76.5%	88.3%	67.2%
Class Axioms	92.7%	89.7%	100%	92.8%	96.4%	91.8%	99.4%
Simple Class Axioms	95.2%	97.6%	100%	95.2%	97.7%	97.9%	99.4%

Table 6: $\overline{oe}_{\mathcal{P}}(\mathcal{O})$ By Source

The Gene Ontology tends to dominate the other sources in Misc due to its extremely large size. It also contains a much higher percentage of complex class axioms than any other ontology we tested, which accounts for the difference in Misc between simple class axioms and class axioms. In LOV there are a considerable number of role axioms. This explains why the expressibility is so much higher for class axioms than all axioms.

In Table 7, we see the range of percent subclass and existential among sources. Anatomy, Ontobee, and Misc all contain medical domain ontologies, which may account for the increase in percent existential if they contain more ontologies in the EL profile. LOV and Hydrography, on the other hand, are expressible with very little subclass at all, and both contain many role axioms. Except for the Anatomy Benchmarks, which is actually only two ontologies so a disproportionately small sample size, it does not appear to be the case that any sources are entirely existential and subclass. Neither are any sources completely lacking the two axiom patterns. When we break the results down by profile in the next section, things will look quite a bit different.

Figure 2 shows the actual counts of each axiom pattern used to calculate expressibility in logarithmic scale. In this chart we can see how sources like Ontobee and Misc do contain some of the less common patterns. They are just so large that smaller sources, like ODPs and benchmarks, tend to have higher percentages. The previously mentioned high percentage of complex class axioms for Misc can be seen in the third column. Also, the two ontology sources with the highest percent expressibility of subclass and existential are Anatomy and Ontobee, both medical type ontology sources. If we move farther down the chart to the less common axiom patterns, the larger ontology sources are less prevalent and now the benchmarks and ODPs start to dominate. The last two axiom patterns were never detected by our program and inverse scoped functional occurred once so the log scale in the chart hides this. For the inverse qualified functional axiom pattern it is conceivable that authors rarely had occasion to write axioms like this. Disjoint classes may seem surprising, however we investigated the evaluation and found that, even though our pattern, $A \sqcap B \sqsubseteq \bot$,

	LOV	Hydrography	Anatomy	Conference	ODP	Ontobee	Misc
Subclass	42.4%	46.2%	66.8%	57.6%	56.4%	60.2%	41.0%
Existential	05.9%	07.4%	33.1%	06.7%	05.7%	26.5%	20.7%
Subclass + Existential	48.3%	53.3%	99.9%	64.3%	62.1%	86.7%	61.7%

Table 7: Percent Subclass and Existential By Source

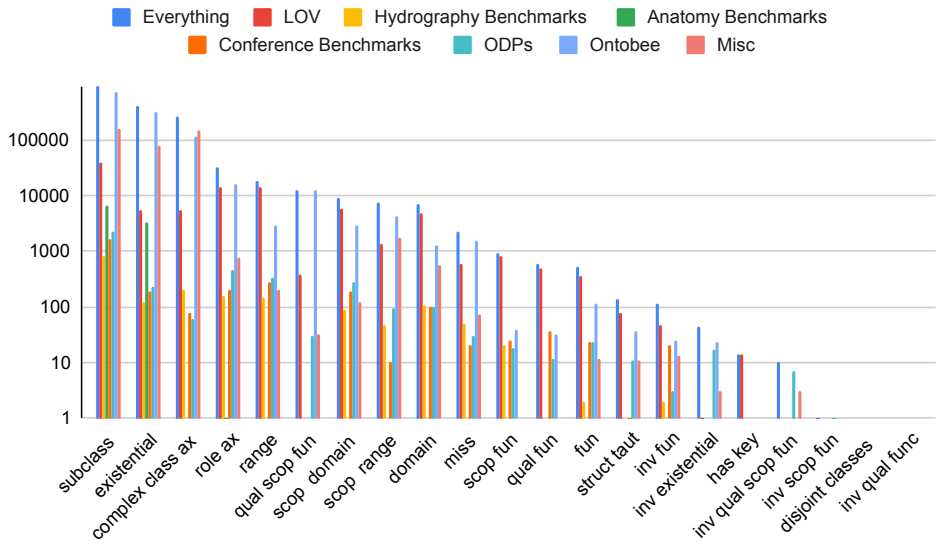


Fig. 2: Axiom Expressibility Counts, Log Scale

is expressible in profiles that do not contain negation, authors are likely using Protégé or the OWLAPI [9] to state disjoint classes axioms which the normalization transforms into subclass axioms containing negation, as defined in the specification. This causes disjoint classes to match the subclass pattern rather than our disjointness pattern, which is not a false negative or a methodological error, but it is technically a misclassification due to conflicting sets of patterns.

4.3 Profile Expressibility

During the analysis we also tested each ontology to see if it was in the OWL profiles EL, QL, RL, or DL, and report the expressibility information for each profile separately. In Tables 8, 9 and 10, the Full column is reproduced from values in Section 4.1 and Table 4 for comparison, since every ontology will be in OWL Full. There were 15 ontologies that could be loaded into the evaluation, but the OWLAPI could not test their profile; these ontologies were not included in the profile results but are included in the Full column. Interestingly, for the EL, QL, and RL profiles we see around a ten percent expressibility boost over the overall result. All three also have perfect expressibility for simple class axioms,

	EL	QL	RL	DL	Full
All Axioms	98.7%	99.7%	97.8%	91.4%	82.2%
Class Axioms	98.8%	100%	100%	92.7%	83.8%
Simple Class Axioms	100%	100%	100%	99.9%	99.8%

Table 8: $\overline{ae}_{\mathcal{P}}(\mathcal{A})$ By Profile

	EL	QL	RL	DL	Full
All Axioms	94.6%	87.7%	81.7%	82.8%	82.9%
Class Axioms	97.5%	97.1%	95.2%	94.4%	92.9%
Simple Class Axioms	98.1%	97.1%	95.2%	97.2%	96.5%

Table 9: $\overline{oe}_{\mathcal{P}}(\mathcal{O})$ By Profile

and nearly perfect expressibility for all class axioms. The expressibility numbers for DL are also slightly higher than the overall numbers, though significantly less so than for the other profiles. The lower values for Full compared to DL are heavily influenced by the Gene Ontology, which was not classified as OWL DL.

Unlike the different sources, where the percent subclass and existential numbers were mostly near the average, we get a much more skewed result when we break the ontologies down by profile in Table 10. EL and DL ontologies seem to be expressible with a similar percentage of subclass axioms as the overall result, though EL has many more existential expressions. QL ontologies, on the other hand, are eighty percent expressible with the simple subclass pattern. And the RL profile ontologies are almost entirely expressible with simple subclass. It is no surprise, then, that EL, QL, and RL ontologies have such high expressibility.

In Table 11, we mark which of our axiom patterns are expressible in each profile with an X symbol, using the OWL 2 Profiles [24] document as a reference. We see that RL expressibility is almost entirely subclass, and the existential pattern is indeed inexpressible in that profile. EL seems to be evenly divided between subclass and existential, which again aligns with the types of statements permitted in the profile. The DL profile allows all the types of expressions and it understandably has a similar result to the overall average.

For the EL profile we also observe a unique result, because our axiom patterns have almost complete overlap with the 4 normal form class axioms defined for $\mathcal{EL}^{++}$ in [1], as shown in Table 12. The only exception is conjunction, which can only match our disjoint classes axiom pattern when the consequent is equal to $\bot$. If we were to define a conjunction axiom pattern, it might be possible to completely express this profile with simple axiom patterns. This could also be done for class axioms in the QL profile, where our axiom patterns could express all simple axioms, and could express any set of QL axioms that was normalized to remove nested quantifiers. Simple class axioms for the RL profile could be expressed in much the same way as EL, missing only conjunction axioms that do not have $\bot$ in the consequent. With the addition of a conjunction pattern

	EL	QL	RL	DL	Full
Subclass	52.5%	78.2%	95.1%	60.9%	54.8%
Existential	46.1%	21.1%	0%	29.2%	23.9%
Subclass + Existential	98.6%	99.3%	95.1%	90.2%	78.8%

Table 10: Percent Subclass and Existential By Profile

	EL	QL	RL	DL		EL	QL	RL	DL
$A \sqsubseteq B$	X	X	X	X	$\top \sqsubseteq \leqslant 1R.\top$				X
$A \sqcap B \sqsubseteq \bot$	X		X	X	$\top \sqsubseteq \leqslant 1R.B$				X
$\exists R.\top \sqsubseteq B$	X	X	X	X	$A \sqsubseteq \leqslant 1R.\top$			X	X
$\exists R.A \sqsubseteq B$	X		X	X	$A \sqsubseteq \leqslant 1R.B$			X	X
$\top \sqsubseteq \forall R.B$				X	$\top \sqsubseteq \leqslant 1R^-.\top$				X
$A \sqsubseteq \forall R.B$			X	X	$\top \sqsubseteq \leqslant 1R^-.B$				X
$A \sqsubseteq \exists R.B$	X	X		X	$A \sqsubseteq \leqslant 1R^-.\top$			X	X
$A \sqsubseteq \exists R^-.B$		X		X	$A \sqsubseteq \leqslant 1R^-.B$			X	X
					$A \sqsubseteq \geqslant 0R.B$				X

Table 11: Profile Axiom Pattern Expressibility

and by normalizing to remove nested quantifiers we could also obtain complete class axiom pattern expressibility for RL.

5 Discussion

Our motivation for this study is that we believe *simple axioms*, specifically those that can be created from simple patterns, are easier for non-logicians to understand and utilize for modeling. This is not to say that complex axioms are unnecessary, indeed they may be the most important. However, if most of OWL could be expressed with simple patterns, as we have shown, this seems a good place to focus our attention when we consider ways to facilitate adoption. Alongside improved comprehension, simple axioms made from patterns come with a number of added benefits: attempting to measure the non-local effects of ontological commitments may be easier, they can be easily and automatically created by tools that allow users to specify statements in a graphical interface without a deep technical understanding of the inner-workings of OWL, and simple axioms often do not require normalization before being input to a reasoner. To support this, we determine the current usage characteristics of axioms in existing ontologies to see if they are expressible with simple axiom patterns, as well as how this relates to different sources and OWL profiles.

Our evaluation is limited in a few ways. We have not yet investigated if limiting an ontology engineer to these axiom patterns would pose additional obstacles, such as in writing complex axioms. There is also no way we are aware of to automatically detect if patterns were used to make an ontology, so we are unable to compare ontologies made without patterns to those that were made with patterns. Finally, our evaluation produces a lower bound. So while it shows

Axiom Pattern	$\mathcal{EL}^{++}$ Normal Class Axiom
$A \sqsubseteq B$	$A \sqsubseteq B$
$A \sqcap B \sqsubseteq \bot$	$A \sqcap B \sqsubseteq C$, when $C = \bot$
$\exists R.\top \sqsubseteq B$	$\exists R.A \sqsubseteq B$, when $A = \top$
$\exists R.A \sqsubseteq B$	$\exists R.A \sqsubseteq B$
$A \sqsubseteq \exists R.B$	$A \sqsubseteq \exists R.B$

Table 12: $\mathcal{EL}^{++}$ Axiom Pattern Expressibility

clearly that axiom patterns can provide sufficient expressibility for most axioms, we do not yet know how much more a more sophisticated evaluation might find.

5.1 Future Work

In the future there are many potential next steps that could build on this study. One approach would be to test different sets of simple axiom patterns and see how the expressibility numbers compare between them. OWLAx was a good basis to create an initial set of simple axiom patterns but there are some obvious common ones that it lacks, for instance conjunction, disjunction, negation, as well as multiple variations on cardinality and role axioms. For the current study we only use simple axioms because there is no clearly defined way to categorize complex axioms, which can be arbitrarily large. It may be interesting to analyze the complex axioms to see if there are any new patterns that can be included.

We also admit that our definition of expressibility is quite simple, intentionally kept this way for clarity. However it may be possible with some more comprehensive statistical tools that a better understanding of axiom pattern expressibility in ontologies is possible. In a future study we may look into different evaluations besides expressibility, perhaps it will be informative to compare.

Additionally, our method normalized many axioms, however it is likely that complex axioms existed in the ontologies we studied that *could* have been normalized but *weren't* because our method only obtained NNF and then split up appropriate conjunction and disjunction axioms. By introducing new terms in the normalization to syntactically split some expressions we might be able to even further increase the expressibility detection capability. Though, as previously mentioned, this would require the addition of new terms, and would be equivalent but also contain more entities, so the comparison would be less obviously appropriate.

6 Conclusion

In this paper we demonstrate that most axioms in OWL ontologies are expressible with a small set of simple axiom patterns. This has implications for how we approach ontology management and development. If ontologies are mostly expressible with simple patterns then focusing on supporting and explaining these types of axiom patterns can lead to easier adoption and maintenance. Complex

axioms will of course always be a part of OWL, but we can improve our ontologies most easily by first making sure that the patterns used to create simple axioms are well understood and used correctly.

Acknowledgement The authors acknowledge partial support from the financial assistance award 70NANB19H094 from U.S. Department of Commerce, National Institute of Standards and Technology and partial support from the National Science Foundation under Grant No. 2033521.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: IJCAI. vol. 5, pp. 364–369 (2005)
2. Blomqvist, E., Hammar, K., Presutti, V.: Engineering Ontologies with Patterns – The eXtreme Design Methodology. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns: Foundations and Applications, Studies on the Semantic Web, vol. 25, chap. 2, pp. 23–50. IOS Press (2016)
3. Falco, R., Gangemi, A., Peroni, S., Shotton, D., Vitali, F.: Modelling OWL ontologies with Graffoo. In: European Semantic Web Conference. pp. 320–325. Springer (2014)
4. Gene Ontology Consortium: The Gene Ontology (GO) database and informatics resource. Nucleic Acids Research **32**(Database-Issue), 258–261 (01 2004). https://doi.org/10.1093/nar/gkh036
5. Glimm, B., Hogan, A., Krötzsch, M., Polleres, A.: Owl: Yet to arrive on the web of data? arXiv preprint arXiv:1202.0984 (2012)
6. Herre, H.: General Formal Ontology (GFO): A foundational ontology for conceptual modelling. In: Theory and applications of ontology: computer applications, pp. 297–345. Springer (2010)
7. Hitzler, P., Krisnadhi, A.: A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR **abs/1808.08433** (2018), `http://arxiv.org/abs/1808.08433`
8. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press (2010)
9. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: Proceedings of the 6th International Conference on OWL: Experiences and Directions – Volume 529. p. 49–58. OWLED'09, CEUR-WS.org, Aachen, DEU (2009)
10. Horridge, M., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Manchester Syntax. W3C Working Group Note (2009)
11. Kang, Y.B., Li, Y.F., Krishnaswamy, S.: Predicting reasoning performance using ontology metrics. In: International Semantic Web Conference. pp. 198–214. Springer (2012)
12. Krisnadhi, A., Hu, Y., Janowicz, K., Hitzler, P., Arko, R.A., Carbotte, S., Chandler, C., Cheatham, M., Fils, D., Finin, T.W., Ji, P., Jones, M.B., Karima, N., Lehnert, K.A., Mickle, A., Narock, T.W., O'Brien, M., Raymond, L., Shepherd, A., Schildhauer, M., Wiebe, P.: The GeoLink Modular Oceanography Ontology. In: Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P.T., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem,

PA, USA, October 11-15, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9367, pp. 301–309. Springer (2015). https://doi.org/10.1007/978-3-319-25010-6_19
13. Lohmann, S., Link, V., Marbach, E., Negru, S.: WebVOWL: Web-based visualization of ontologies. In: International Conference on Knowledge Engineering and Knowledge Management. pp. 154–158. Springer (2014)
14. Matentzoglu, N., Bail, S., Parsia, B.: A snapshot of the owl web. In: International Semantic Web Conference. pp. 331–346. Springer (2013)
15. Parsia, B., Patel-Schneider, P., Motik, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C recommendation, W3C (Dec 2012), http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/
16. Sarker, M.K., Krisnadhi, A., Carral, D., Hitzler, P.: Rule-based OWL modeling with ROWLTab Protégé plugin. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) The Semantic Web – 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10249, pp. 419–433 (2017)
17. Sarker, M.K., Krisnadhi, A.A., Hitzler, P.: OWLAx: A Protégé plugin to support ontology axiomatization through diagramming. In: Kawamura, T., Paulheim, H. (eds.) Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016. CEUR Workshop Proceedings, vol. 1690 (2016)
18. Shimizu, C.: Towards a comprehensive modular ontology IDE and tool suite. In: Kirrane, S., Kagal, L. (eds.) Proceedings of the Doctoral Consortium at ISWC 2018 co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th to12th, 2018. CEUR Workshop Proceedings, vol. 2181, pp. 65–72 (2018)
19. Shimizu, C., Hammar, K., Hitzler, P.: Modular graphical modeling evaluated. In: Proceedings of ESWC (2020), to appear
20. Shimizu, C., Hirt, Q., Hitzler, P.: MODL: A Modular Ontology Design Library. In: Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019). CEUR Workshop Proceedings, vol. 2459, pp. 47–58 (2019)
21. Shimizu, C., Hitzler, P., Hirt, Q., Shiell, A., Gonzalez, S., Foley, C., Rehberger, D., Watrall, E., Hawthorne, W., Tarr, D., Carty, R., Mixter, J.: The Enslaved Ontology 1.0: People of the historic slave trade. Tech. rep., Michigan State University, East Lansing, Michigan (April 2019)
22. Vandenbussche, P., Atemezing, G., Poveda-Villalón, M., Vatant, B.: Linked open vocabularies (LOV): A gateway to reusable semantic vocabularies on the web. Semantic Web **8**(3), 437–452 (2017). https://doi.org/10.3233/SW-160213, `https://doi.org/10.3233/SW-160213`
23. Wang, T.D., Parsia, B., Hendler, J.: A survey of the web ontology landscape. In: International Semantic Web Conference. pp. 682–694. Springer (2006)
24. Wu, Z., Fokoue, A., Grau, B.C., Horrocks, I., Motik, B.: OWL 2 Web Ontology Language Profiles (Second Edition). W3C recommendation, W3C (Dec 2012), http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/
25. Xiang, Z., Mungall, C., Ruttenberg, A., He, Y.: Ontobee: A linked data server and browser for ontology terms. In: ICBO (2011)
26. Zhang, H., Li, Y.F., Tan, H.B.K.: Measuring design complexity of semantic web ontologies. Journal of Systems and Software **83**(5), 803–814 (2010)

A Domain Ontology for Task Instructions

Aaron Eberhart✉[1][0000−0003−3007−5460], Cogan Shimizu[1][0000−0003−4283−8701], Christopher Stevens[2], Pascal Hitzler[1][0000−0001−6192−3472], Christopher W. Myers[2][0000−0003−0556−5935], and Benji Maruyama[3]

[1] DaSe Lab, Kansas State University, Manhattan, KS, USA
[2] Air Force Research Laboratory, Wright-Patterson AFB, OH, USA
[3] Air Force Research Laboratory, Materials & Manufacturing Directorate, Wright-Patterson AFB, OH, USA
{aaroneberhart, coganmshimizu, hitzler}@ksu.edu, {christopher.myers.29, christopher.stevens.28, benji.maruyama}@us.af.mil

Abstract. Knowledge graphs and ontologies represent information in a variety of different applications. One use case, the Intelligence, Surveillance, & Reconnaissance: Mutli-Attribute Task Battery (ISR-MATB), comes from Cognitive Science, where researchers use interdisciplinary methods to understand the mind and cognition. The ISR-MATB is a set of tasks that a cognitive or human agent perform which test visual, auditory, and memory capabilities. An ontology can represent a cognitive agent's background knowledge of the task it was instructed to perform and act as an interchange format between different Cognitive Agent tasks similar to ISR-MATB. We present several modular patterns for representing ISR-MATB task instructions, as well as a unified diagram that links them together.

1 Introduction

Knowledge graphs facilitate data integration across highly heterogeneous sources in a semantically useful way. Knowledge graphs may be equipped with a schema, frequently an ontology, that combines the associative power of the knowledge graph with the semantics of the ontology. Due to this, they are uniquely suited to support research in cognitive science, where it is often necessary to incorporate information from fields like computer science, psychology, neuroscience, philosophy, and more.

Cognitive agents are a sub-field of cognitive science and an application of the more broad study of cognitive architectures. Cognitive architectures, like ACT-R [1] for example, are an approach to understanding intelligent behavior and cognition that grew out of the idea of Unified Theories of Cognition [8]. These systems have their roots in AI production systems and some types use rules-based cognition. Many in Computer Science are familiar with inductive themes from a different type, called Connectionism, due to its historic ties with artificial neural networks. Symbolic cognitive architectures, by contrast, are less widely known outside of cognitive science, and are abstracted and explicit like logic programming.

Both ontologies and cognitive architectures deal with symbolic knowledge. Symbolic cognitive architectures typically focus on the plausibility of knowledge and the way in which that knowledge is translated into human behavior within a specific task. Ontologies offer a set of robust mechanisms for reasoning over complex knowledge bases and could help cognitive architectures adapt to tasks in novel environments. One way the two may be integrated is by leveraging the ontology to reduce the specificity of a cognitive agent.

In general, cognitive agents are often specialized, or *differentiated*, to perform a specific task or set of tasks. An *undifferentiated* agent is one that has no specialization. The purpose of such an agent is to be adaptable to new tasks as needed. As part of initial work to develop such an undifferentiated cognitive agent, we have developed a modular ontology that captures instructions for a specific cognitive agent task called ISR-MATB. We discuss this platform in more detail in the next section.

Currently, the ontology supports the memory of a cognitive agent by adding structure to its knowledge and providing new varieties of query-like recall. And due to design methodology used during the modeling process, the ontology is general enough that it could model other cognitive agent experiments, which could then be evaluated against each other in a structured way. This allows the ontology to act as an invaluable interchange format between researchers developing cognitive agents.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the use-case: ISR-MATB. Section 3 provides an in-depth examination of the ontology. Finally, in Section 4, we briefly conclude and discuss next steps.

2 ISR-MATB

ISR-MATB is a series of cognitive tasks that could be completed by a Cognitive Agent or a human [3]. A trial starts with one very simple task, the evaluation then branches into two sub-tasks that relate back to the first task. After the two sub-tasks are complete the agent completes one final task requiring integration of remembered information from all previous tasks. The final task is made more difficult by the possibility of incorrect feedback as the agent learns. ISR-MATB is intended to be repeated for a fixed time so that researchers can observe changes in the agent's response time and develop better computational cognitive agents.

2.1 Psychomotor Vigilance Test

The Psychomotor Vigilance Test is one of the more basic cognitive tasks [2]. In this task, there is an area of the screen where a letter could appear. The letter will be drawn with a specific color. When the letter does appear an agent must press a button that acknowledges they have seen it. If the agent pushes the button too soon a false start is recorded and the task continues normally. If too much time passes before the agent pushes the button then the task will continue

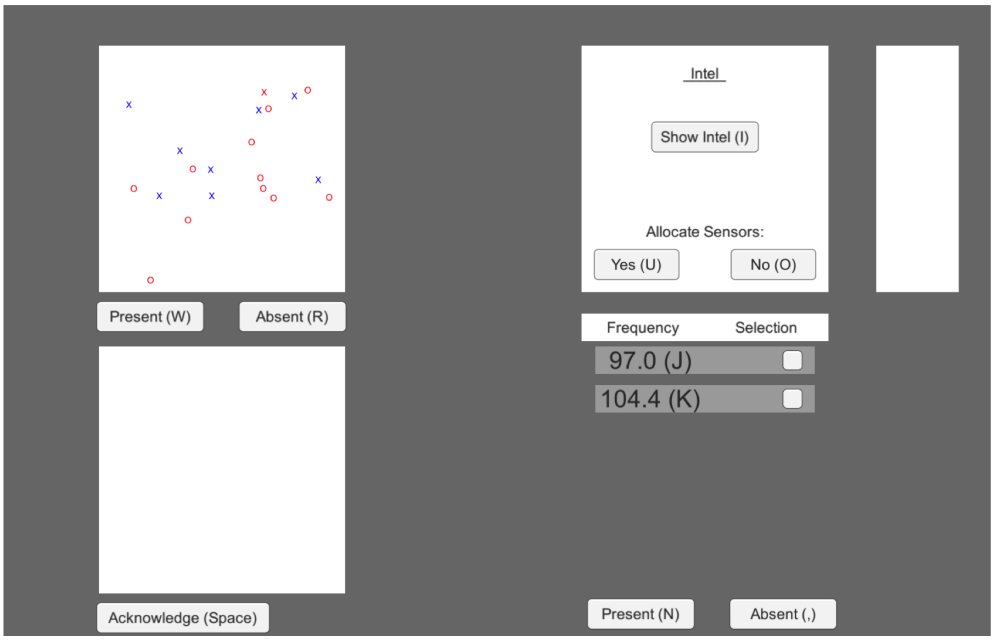


Fig. 1: An example depicting the four ISR-MATB tasks in a single interface.

with the letter unacknowledged. The next two tasks reference this letter and color, so the agent is instructed to remember them.

2.2 Visual Search

The Visual Search task requires that the agent determine if the letter they remember is among a group of many letters that appear on the screen [10]. The other letters are distractors, and may be the same letter as the target with a different color, or the same color as the target with a different letter, or both color and letter different. The target may or may not appear among the distractors, and never appears more than once. The agent pushes a button to indicate whether the the letter is present or absent.

2.3 Auditory Search

The Auditory Search task is very similar to the Visual Search, except of course that the agent must listen instead of look. In this task there are between one and four audio messages that each include a spoken color and letter. If one of the messages is the same as as the letter and color from the first task the agent pushes a button to indicate that it is present, otherwise they indicate that it is absent.

2.4 Decision Making

The final task, Decision Making, requires agents to infer a relationship between the outcomes of the Visual and Auditory Search tasks together with a new binary piece of information called "Intelligence" that appears after choosing whether to hypothetically allocate sensors or not. The rule the agent must guess is not too hard, but it is complex enough that it must be learned by trial-and-error over multiple attempts. Learning the rule is made more difficult by the unlikely but not impossible event that the program responds incorrectly even when a correct answer is given. Responding 'yes' or 'no' to this sub-task ends one ISR-MATB trial.

3 Ontology Description

In this section we present the Instruction Ontology, a domain ontology built for use with the ISR-MATB experiment platform. This ontology was produced by following the Modular Ontology Modeling (MOM) methodology, outlined in [7,6] MOM is designed to ensure the high quality and reusability of the resulting ontology, both in terms of scope and in terms of granularity, which is a desired outcome.

The ontology consists of six modules: ISR-MATB Experiment, Instruction, SituationDescription, ItemRole, Action, and Affordance. For each module, we describe its purpose, provide a schema diagram,[4] and state its axiomatization in both description logic syntax and natural language. The OWL file for this ontology can be found online[5] as well as the official documentation.[6] Figure 5 shows the schema diagram for the entire ontology.

3.1 ISR-MATB Experiment.

The ISR-MATB Experiment module is the core module for the ontology. The two main classes are ISR-MATB Experiment and ISR-MATB Task. As noted in Section 2, an experiment consists of up to four tasks that may require that information be carried between them, where each Task resides in a specific quadrant of the interface. Each Task provides roles to different Items, as well as a set of Instructions for the agent to carry out. We discuss these classes in more detail in their respective Module sections. The schema diagram for this module is shown in Figure 2c.

[4] A schema diagram is an informal, but intuitive way for conveying information about the structure and contents of an ontology. We use a consistent visual syntax for convenience, detailed in Figure 2.

[5] See `https://raw.githubusercontent.com/undiffagents/uagent/develop/ontology/uagent.owl`.

[6] See `https://daselab.cs.ksu.edu/content/domain-ontology-instruction`

Axiomatization:

$$\top \sqsubseteq \forall \mathsf{affords.Affordance} \quad (1)$$
$$\mathsf{ISR\text{-}MATBTask} \sqsubseteq \geq 1\ \mathsf{hasInstruction.Instruction} \quad (2)$$
$$\mathsf{ISR\text{-}MATBExperiment} \sqsubseteq \leq 4\ \mathsf{hasTask.ISR\text{-}MATBTask} \quad (3)$$
$$\top \sqsubseteq \forall \mathsf{hasLocation.Location} \quad (4)$$
$$\top \sqsubseteq \forall \mathsf{hasName.xsd{:}string} \quad (5)$$
$$\mathsf{ISR\text{-}MATBTask} \sqsubseteq = 1\ \mathsf{hasName.xsd{:}string} \quad (6)$$
$$\mathsf{ISR\text{-}MATBTask} \sqsubseteq \forall \mathsf{providesRole.ItemRole} \quad (7)$$
$$\mathsf{ISR\text{-}MATBTask} \sqsubseteq \forall \mathsf{informs.ISR\text{-}MATBTask} \quad (8)$$

Explanation of axioms above:

1. Range. The range of affords is Affordance.
2. Minimum Cardinality. An ISR-MATBTask has at least one Instruction.
3. Maximum Cardinality. An ISR-MATBExperiment consists of at most four ISR-MATBTasks.
4. Range. The range of hasLocation is Location.
5. Range. The range of hasName is xsd:string.
6. Scoped Range. The range of providesRole is ItemRole when the domain is ISR-MATBTask.
7. Scoped Range. The range of informs is ISR-MATBTask when the domain is ISR-MATBTask.

3.2 Action

The Action module is an instantiation of the Explicit Typing meta-pattern described in [9].[7]

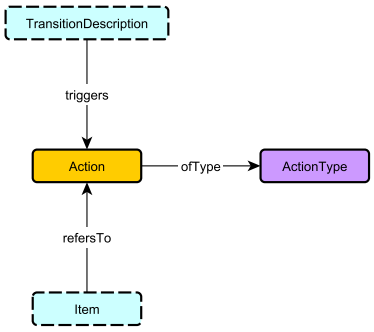
In this case, we use a class, ActionType, to represent a controlled vocabulary. We believe that using a controlled vocabulary to represent this type information is less invasive to the ontology. This way, adding or removing types of actions from the controlled vocabulary does not actually change the ontology. Some instances of the controlled vocabulary are listed in Figure 5.

An Action, in this context, is the physical, actual action that takes place to transition between different states of the experiment, e.g. 'the action of clicking a button.' The schema diagram for this module is shown in Figure 2a.

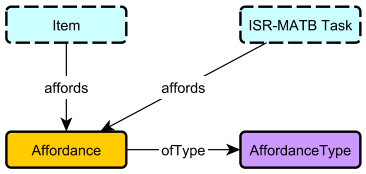
Axiomatization:

$$\mathsf{Action} \sqsubseteq = 1 \mathsf{ofType.ActionType} \quad (1)$$

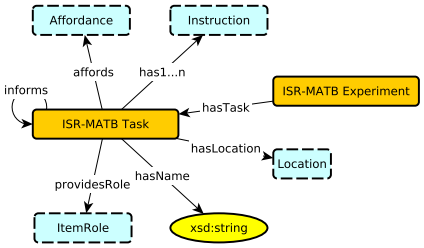
[7] [9] is a modular ontology design library; it contains a set of frequently used patterns and respective documentation.



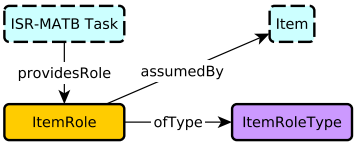
(a) The schema diagram for the Action module.



(b) The schema diagram for the Affordance module.



(c) The schema diagram for the ISR-MATB Experiment module.



(d) The schema diagram for the ItemRole module.

Fig. 2: Orange boxes are classes and indicate that they are central to the diagram. Blue dashed boxes indicate a reference to another diagram, pattern, or module. Gray frames with a dashed outline contain modules. Arrows depict relations and open arrows represent subclass relations. Yellow ovals indicate data types (and necessarily, arrows pointing to a datatype are data properties). Finally, purple boxes represent controlled vocabularies. That is, they represent a controlled set of IRIs that are of that type.

Explanation of axioms above:

1. Exact Cardinality. An Action has exactly one ActionType.

3.3 Affordance

The Affordance module is also instantiated from the Explicit Typing meta-pattern, explained in more detail in Section 3.2 and [9]. An Affordance is essentially some quality of an Item that indicates that "something" may be done with it. Familiar examples might include *clickable* buttons or text highlighted in blue (perhaps indicating that it's a hyperlink). Instances of the AffordanceType can be found in Figure 5. The schema diagram for this module is shown in Figure 2b.

Axiomatization:

$$\mathsf{Affordance} \sqsubseteq =1\mathsf{hasAffordanceType.AffordanceType} \quad (1)$$

Explanation of axioms above:

1. Exact cardinality. An Affordance has exactly one AffordanceType.

3.4 ItemRole

The ItemRole module is an instantiation of the AgentRole pattern, which may also be found in [9]. We also equip it with an explicit type, in the same manner as Action and Affordance.

Each ISR-MATB Task may provide roles to Items. That is, certain items may be a target or distractor, but not always. This allows us to assign certain roles to items that may, if they were qualities, be ontologically disjoint. The schema diagram for this module is shown in Figure 2d.

Axiomatization:

$$\mathsf{ISR\text{-}MATBTask} \sqsubseteq \forall \mathsf{providesRole.ItemRole} \quad (1)$$
$$\top \sqsubseteq \forall \mathsf{hasItemRoleType.ItemRoleType} \quad (2)$$
$$\mathsf{ItemRole} \sqsubseteq \forall \mathsf{assumedBy.Item} \quad (3)$$
$$\mathsf{ItemRole} \sqsubseteq \exists \mathsf{assumedBy.Item} \quad (4)$$

Explanation of axioms above:

1. Scoped Range. The range of providesRole is ItemRole when the domain is ISR-MATBTask.
2. Range. The range of hasItemRoleType is ItemRoleType.
3. Scoped Range. ItemRoles are assumedBy Items.
4. Existential. Every ItemRole is assumedBy an Item.

3.5 SituationDescription

For this module, we opted to use the Situation and Description approach. We chose to use this conceptualization due to the non-linear nature of the instructions.[8] That is, an ISR-MATB Task is not a sequence of instructions, but a collection of directions or descriptions.

An Instruction, is a description of a way to transition between two states. In order to follow out an instruction the state described in the the pre-SituationDescription would need to be met. Following through would result in a new state, the Post-Situtation Description.

[8] For a deeper discussion on Descriptions, Situations, and Plans, see [4].

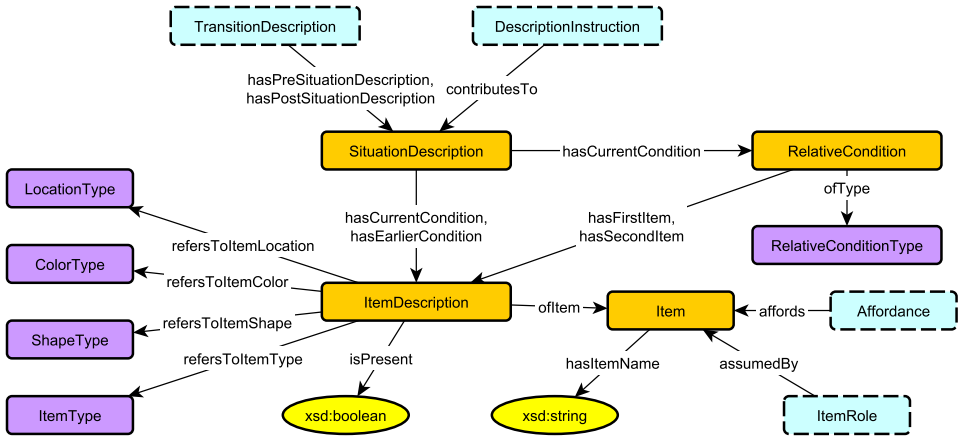


Fig. 3: The schema diagram for the SchemaDiagram module. Color and shape usage is the same as in previous diagrams.

Furthermore, the SituationDescription will indicate the presence, or absence, of an item, as well as its description. Descriptions, in this case, are relegated to controlled vocabularies in the same manner as Affordance or Action. We call this an ItemDescription because it is inherent to the Instruction and not the Item, itself.

The schema diagram for this module is shown in Figure 3.

Axiomatization:

$$\mathsf{SituationDescription} \sqsubseteq \forall \mathsf{hasCurrentCondition}.(\mathsf{RelativeCondition} \sqcup \mathsf{ItemDescription}) \tag{1}$$

$$\mathsf{SituationDescription} \sqsubseteq \forall \mathsf{hasEarlierCondition}.\mathsf{ItemDescription} \tag{2}$$

$$\top \sqsubseteq \forall \mathsf{hasRelativeConditionType}.\mathsf{RelativeConditionType} \tag{3}$$

$$\mathsf{RelativeCondition} \sqsubseteq \forall \mathsf{hasFirstItem}.\mathsf{ItemDescription} \tag{4}$$

$$\mathsf{RelativeCondition} \sqsubseteq \forall \mathsf{hasSecondItem}.\mathsf{ItemDescription} \tag{5}$$

$$\mathsf{ItemDescription} \sqsubseteq \forall \mathsf{ofItem}.\mathsf{Item} \tag{6}$$

$$\mathsf{ItemDescription} \sqsubseteq =1\ \mathsf{isPresent}.\mathsf{xsd{:}boolean} \tag{7}$$

$$\top \sqsubseteq \forall \mathsf{refersToItemLocation}.\mathsf{LocationType} \tag{8}$$

$$\top \sqsubseteq \forall \mathsf{refersToItemColor}.\mathsf{ColorType} \tag{9}$$

$$\top \sqsubseteq \forall \mathsf{refersToShapeType}.\mathsf{ShapeType} \tag{10}$$

$$\top \sqsubseteq \forall \mathsf{refersToItemType}.\mathsf{ItemType} \tag{11}$$

$$\mathsf{ItemDescription} \sqsubseteq \geq 0\ \mathsf{refersToItemLocation}.\mathsf{LocationType} \tag{12}$$

$$\mathsf{ItemDescription} \sqsubseteq \geq 0\ \mathsf{refersToItemColor}.\mathsf{ColorType} \tag{13}$$

$$\text{ItemDescription} \sqsubseteq \geq 0\ \text{refersToItemShape.ShapeType} \quad (14)$$

$$\text{ItemDescription} \sqsubseteq \geq 0\ \text{refersToItemType.ItemType} \quad (15)$$

$$\top \sqsubseteq \forall \text{hasItemName.xsd:string} \quad (16)$$

$$\exists \text{hasItemName.}\top \sqsubseteq \text{Item} \quad (17)$$

Explanation of axioms above:

1. Scoped Range. The range of hasCurrentCondition is a RelativeCondition or ItemDescription when the domain is SituationDescription.
2. Scoped Range. The range of hasEarlierCondition is ItemDescription when the domain is SituationDescription.
3. Range. The range of hasRelativeConditionType is RelativeConditionType.
4. Scoped Range. The range of hasFirstItem is ItemDescription when the domain is RelativeCondition.
5. Scoped Range. The range of hasSecondItem is ItemDescription when the domain is RelativeCondition.
6. Scoped Range. The range of ofItem is Item when the domain is ItemDescription.
7. Scoped Range. An ItemDescription has exactly one Boolean flag indicating whether or not it is present.
8. Range. The range of refersToItemLocation is LocationType.
9. Range. The range of refersToItemColor is ColorType.
10. Range. The range of refersToItemShape is ShapeType.
11. Range. The range of refersToItemType is ItemType.
12. Structural Tautology. An ItemDescription may refer to a LocationType.
13. Structural Tautology. An ItemDescription may refer to a ColorType.
14. Structural Tautology. An ItemDescription may refer to a ShapeType.
15. Structural Tautology. An ItemDescription may refer to an ItemType.
16. Range. The range of hasItemName is xsd:string.
17. Domain Restriction. The domain of hasItemName is restricted to Items.

3.6 Instruction

Instructions are the atomic units of a task. They come in two varieties: descriptions and actions. The former are instructions that are prescriptive or descriptive. They are statements that indicate information about the environment or the task. They may, in natural language, take such form as "There is a button named 'Present'." The latter type of instruction instructs when or where to do something. For example, "Press the button if a high-pitched tone is heard." An Action-Instruction prescribes some transition between descriptions of situations, whereas Description-Instructions directly contribute to said SituationDescription. The module also uses a data property to capture the natural language formulation of the Instruction. The schema diagram for this module is shown in Figure 4.

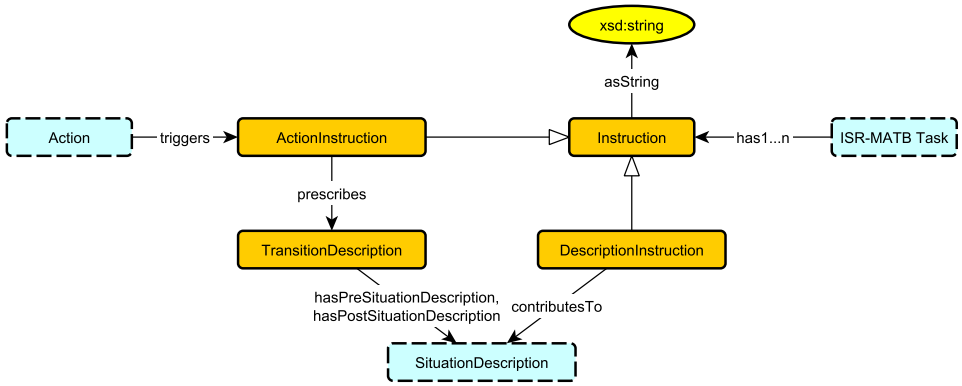


Fig. 4: The schema diagram for the Instruction module. Color and shape usage is the same as in previous diagrams.

Axiomatization:

$$\mathsf{ActionInstruction} \sqsubseteq \mathsf{Instruction} \tag{1}$$

$$\mathsf{ActionInstruction} \sqsubseteq \forall \mathsf{prescribes}.\mathsf{TransitionDescription} \tag{2}$$

$$\top \sqsubseteq \forall \mathsf{asString}.\mathsf{xsd{:}string} \tag{3}$$

$$\mathsf{Instruction} \sqsubseteq \geq 0\ \mathsf{asString}.\mathsf{xsd{:}string} \tag{4}$$

$$\mathsf{DescriptionInstruction} \sqsubseteq \mathsf{Instruction} \tag{5}$$

$$\mathsf{DescriptionInstruction} \sqsubseteq \forall \mathsf{contributesTo}.\mathsf{SituationDescription} \tag{6}$$

$$\top \sqsubseteq \forall \mathsf{hasPreSituationDescription}.\mathsf{SituationDescription} \tag{7}$$

$$\top \sqsubseteq \forall \mathsf{hasPostSituationDescription}.\mathsf{SituationDescription} \tag{8}$$

Explanation of axioms above:

1. Subclass. Every ActionInstruction is an Instruction.
2. Scoped Range. The range of prescribes is TransitionDescription when the domain is ActionInstruction.
3. Range. The range of asString is xsd:string.
4. Structural Tautology. An Instruction may have a string representation.
5. Subclass. Every DescriptionInstruction is an Instruction.
6. Scoped Range. The range of contributesTo is SituationDescription when the domain is DescriptionInstruction.
7. Range. The range of hasPreSituationDescription is SituationDescription.
8. Range. The range of hasPostSituationDescription is SituationDescription.

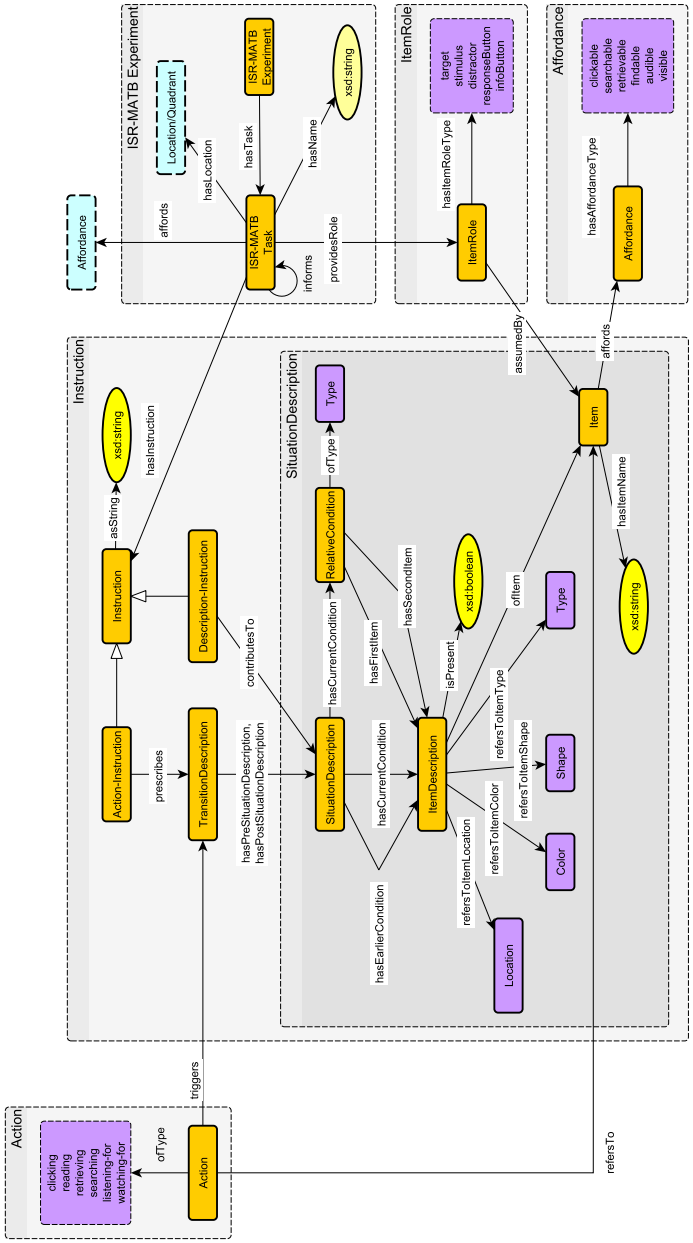


Fig. 5: The schema diagram for the entire ontology. Note that the SituationDescription module is nested in the Instruction Module. Color and shape usage is the same as in previous diagrams.

4 Conclusion

In this paper we have presented an ontology for modeling the ISR-MATB cognitive agent task instructions. This ontology can be used, as we have, to directly support the memory of a cognitive agent performing tasks. It also could support experiment design, irrespective of any agent, by providing a structured basis for evaluating similar tasks. The modular structure facilitates adapting the ontology to other use cases and scenarios by replacing or adapting the existing modules. It is also possible to create new modules from the referenced patterns via template-based instantiation [5].

4.1 Future Work

In the future we plan to extend this ontology so that it can support a fully undifferentiated agent. This will include tasks like ISR-MATB, but also many others that could be very different. One such task is supporting materials science research that uses the Autonomous Research System (ARES) framework. An undifferentiated cognitive agent could operate a robotic system that performs research, using software like ARES, saving materials researchers hours of potentially hazardous lab work.

Acknowledgement This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386.

References

1. Anderson, J.R.: How can the human mind occur in the physical universe? Oxford University Press (2007)
2. Dinges, D.F., Powell, J.W.: Microcomputer analyses of performance on a portable, simple visual rt task during sustained operations. Behavior research methods, instruments, & computers 17(6), 652–655 (1985)
3. Frame, M., Lopez, J., Myers, C., Stevens, C., Estepp, J., Boydstun, A.: Development of an autonomous management system for human-machine teaming with multiple interdependent tasks. In: Presented to the Annual Meeting of the Psychonomic Society Conference, Montreal, QC, Canada, November 2019 (2019)
4. Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE – OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Lecture Notes in Computer Science, vol. 2888, pp. 689–706. Springer (2003)
5. Hammar, K., Presutti, V.: Template-based content ODP instantiation. In: Hammar, K., Hitzler, P., Krisnadhi, A., Lawrynowicz, A., Nuzzolese, A.G., Solanki, M. (eds.) Advances in Ontology Design and Patterns [revised and extended versions of the papers presented at the 7th edition of the Workshop on Ontology and Semantic Web Patterns, WOP@ISWC 2016, Kobe, Japan, 18th October 2016]. Studies on the Semantic Web, vol. 32, pp. 1–13. IOS Press (2016), https://doi.org/10.3233/978-1-61499-826-6-1

6. Hitzler, P., Krisnadhi, A.: A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR abs/1808.08433 (2018), http://arxiv.org/abs/1808.08433
7. Krisnadhi, A., Hitzler, P.: Modeling with ontology design patterns: Chess games as a worked example. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 3–21. IOS Press (2016)
8. Newell, A.: Unified theories of cognition. Harvard University Press (1994)
9. Shimizu, C., Hirt, Q., Hitzler, P.: MODL: A modular ontology design library. In: WOP@ISWC. CEUR Workshop Proceedings, vol. 2459, pp. 47–58. CEUR-WS.org (2019)
10. Treisman, A.M., Gelade, G.: A feature-integration theory of attention. Cognitive psychology 12(1), 97–136 (1980)