# LOGIC-BASED NLP AND ITS USES

Caleb Young Alexander

**A BACKGROUND DOCUMENT**

Presented to the Faculty of Miami University in partial
fulfillment of the requirements
for the degree of

Master of Science

Department of Computer Science & Software Engineering

The Graduate School
Miami University
Oxford, Ohio

2023

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction to Logic Programming

This chapter will serve as a brief introduction to the areas of logic programming relevant to the proposed research. For further reading, the book **Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach** by Michael Gelfond and Yulia Kahl is heavily recommended. Indeed, much of the information from this chapter will be drawn from the works of Michael Gelfond.

## 1.1 Logic Programming Basics and ASP

The proposed research's main field of focus is logic programming and knowledge representation. As such, an explanation of the field is prudent. Logic programming, is a declarative approach to programming built on top of first-order logic's. Traditional programming uses an algorithmic approach rather than the declarative approach that logic programming prefers. An algorithmic program describes the actions that should be taken by a machine to reach a desired end state. This traditional approach resembles a recipe, in that it describes the ingredients needed (memory, computing power etc.) and the steps to be taken (declare an object, operate on it, etc.) so that some outcome is reached. In contrast, declarative programming lists existing objects and the relations between them [4]. More formally, a logic program describes a map from a set (the set of given objects) to itself.

For a clearer example of logic programming in practice, an example taken from [4] will be used. Consider the following logic program,

| | |
|---|---|
| $father(john, sam).$ | john is the father of sam |
| $mother(alice, sam).$ | alice is the mother of sam |
| $gender\_of(john, male).$ | john is male |
| $gender\_of(sam, male).$ | same is male |
| $gender\_of(alice, female).$ | alice is female |
| | |
| $parent(X, Y) : -father(X, Y)$ | If X is a father of Y then X is a parent of Y |
| $parent(X, Y) : -mother(X, Y)$ | If X is a mother of Y then X is a mother of Y |
| $child(X, Y) : -parent(Y, X)$ | If Y is a parent of X then X is a child of Y |

To the right of each line is how it would be read in English. Each of the lines in this logic program

are called predicates because they describe some relation between objects. In this case the object set, $O$, is given as $O = \{john, sam, alice, male, female\}$. Each object, $Object$, can be related to another object, $Object_2$, in the following ways: $Object$ can be a parent of $Object_2$ or vice-versa, $Object$ can be the gender of $Object_2$ or vice-versa, $Object$ can be the mother of $Object_2$ or vice-versa, $Object$ can be the father of $Object_2$ or vice-versa, and $Object$ can be the child of $Object_2$ or vice-versa. Something important to remember about predicates is that they are binary. This means that an object is either related to another in a certain way or it is not; there are no third options.

Generally, a user interacts with a logic program by way of queries. For instance, a user could enter the query $?mother(X, sam)$ which would return the $X$ which is john's mother. In this case the object returned would be $alice$. As more is added to a logic program, it becomes more useful to a user because consequences of relations can be relayed back to the user. Two easy additions to this program could be,

$father(X, Y) : -parent(X, Y), gender\_of(X, male).$
If X is the parent of Y and X is male then X is the father of Y

$mother(X, Y) : -parent(X, Y), gender\_of(X, female).$
If X is the parent of Y and X is female then X is the mother of Y.

Now users are able to find mother and father relations by knowing a parent relation and the gender of the parent. Before, if all that was specified was the parent relation and gender relation, the logic program would have no way of deducing a mother or father relationship.

There are many logic programming languages however, the focus of this paper will be Answer Set Prolog (ASP) and similar languages. The fundamental property of ASP is it's signature, $\Sigma$, a four-tuple of disjoint sets $\Sigma = \langle O, F, P, V \rangle$. The sets are $O$ the set of objects, $F$ the set of functions, $P$ the set of predicates, $V$ the set of variables. Any term in an ASP program can be fit into one of those sets and together, they uniquely denote any program. An answer set is the set of valid consequences of an ASP program. In the program given before a term like "John is the parent of Sam" would be contained in the answer set because it is a valid consequence of the rules described [5]. Note that John was never explicitly said to be the parent of Sam, however it is a logical consequence of the written statements. Therefore, the answer set of an ASP program contains the set of possible answers to valid queries.

An important property of logic programs to remember going forward is that logic programs, like any first-order logic, is entirely deterministic. This means, that all conclusions follow from the predicates and there is no room for ambiguity. This property will be important when comparing logic based natural language processing and statistical natural language processing.

## 1.2 AOPL and Specialized Logic Programming Languages

Similar to extensions of algorithmic languages, ASP and other logic programming languages are perfectly capable of being extended. Specialized tasks often benefit from specialized languages.

One such language is the Authorization and Obligation Policy Language (AOPL). A key concern in the field of AI is controlling what actions any intelligent machine is capable of performing. Thus the need for a specialized and machine readable language for defining permitted actions arises. Many domain models made for the purpose of logic programs are static, for the reason that such domains are easier to model. However, most useful models will be dynamin. Such a domain, from a logic programming perspective, is modeled through states and the transitions between said states [6]. AOPL is a language specialized for dynamic domains in that it describes an agent, states, and transitions between states. An agent is the focus of the program which is capable of taking actions. A state is simply a set of objects and the current relations between those objects. Finally a transition is a mapping from one state to another [7]. Typically, a transition will be executed as an action of the agent and will transition the current state into a new state. The content of an AOPL program will look like a description of which transitions $\tau$ are permitted in which states $S$. An example taken from [7] is given below.

| | |
|---|---|
| $permitted(\tau)$ if $cond$ | The transition $\tau$ is permitted if $cond$ is true |
| $\neg permitted(\tau)$ if $cond$ | The transition $\tau$ is not permitted if $cond$ is true |
| $d_1$: normally $permitted(\tau)$ if $cond$ | The transition $\tau$ is normally permitted if $cond$ is true |
| $d_2$: normally $\neg permitted(\tau)$ if $cond$ | The transition $\tau$ is normally not permitted if $cond$ is true |
| $prefer(d_1, d_2)$ | Prefer the rule $d_1$ over $d_2$ if both are true. |

A few of the special features above demand some further explanation. First, the above policy is an authorization policy because it describes which actions are $permitted$ but makes no note of any actions that *must* be taken. An agent, given the policy above, *may* choose to take any $permitted$ action or none of them. Another important feature is the normally keyword. This keyword allows for permissions to be restricted to very specific cases. If an action is described as normally not permitted but permitted in one scenario, that is not a contradiction. Instead the more specific condition takes precedence (i.e the condition without normally). Similarly, through the $prefer$ keyword, even statements that both have normally in them can disagree as long as one has been stated as preferred. This approach allows for a high level of flexibility in our programs and cuts down on the need to describe every scenario in which an action is $permitted$ or $\neg permitted$.

This explains the Authorization part of AOPL, next we discuss the Obligation part. In contrast to an authorization policy, an obligation policy describes the conditions in which an action *must* be taken. For instance the example taken from [7] below is an obligation policy.

| | |
|---|---|
| $obl(\tau)$ if $cond$ | The transition $\tau$ is obligated if $cond$ is true |
| $\neg obl(\tau)$ if $cond$ | The transition $\tau$ is not obligated if $cond$ is true |
| $d_1$: normally $obl(\tau)$ if $cond$ | The transition $\tau$ is normally obligated if $cond$ is true |
| $d_2$: normally $\neg obl(\tau)$ if $cond$ | The transition $\tau$ is normally not obligated if $cond$ is true |
| $prefer(d_1, d_2)$ | Prefer the rule $d_1$ over $d_2$ if both are true. |

Much of the above is identical to the authorization approach. The only real difference being the

use of the *obl* keyword rather than *permitted*. The real power of AOPL only becomes clear when using both in a policy. Using both allows for an author to create an interconnected and broad rule set for an agent quickly and without the need to specify every scenario when an action might be obligated, permitted, or both.

The proposed research could dramatically benefit AOPL and other specialized languages in a few ways. If the end goal of a user is to specify a policy in AOPL, then it would be much easier to do so in English and translate that English to AOPL rather than learning the syntax of AOPL from scratch. Similarly, if the end goal is to write an English policy, then translating back and forth from AOPL to English can provide huge benefits in terms of policy analysis. For starters, contradictions and unclear language can be caught easily just by checking if the program will compile as valid. Next, a few simple queries could check for tautologies or extraneous conditions. Ultimately, the proposed research would cut down on the complexity of using AOPL and other specialized logic programming languages.

## 1.3 Logic Programming as NLP

The bulk of the proposed research will focus on the overlap between logic programming and natural language processing (NLP). Thus, this section will provide an overview of how NLP is related with logic programming. In some sense, logic programming is a very natural way of approaching the task of NLP. Written natural language rests upon linguistic frameworks; large sets of rules which describe the functions of words and phrases in given contexts. Generally, this is called a knowledge base [8] and the creation of effective large knowledge bases is one of the core issues in the field of logic programming. This heavily resembles a formal logic system. The main trouble, comes from the sheer number of rules and exceptions in linguistic frameworks. Linguists have for centuries tried to define complete linguistic systems which include all of English, but the very nature of written natural language defies complete classification. There are, of course, general rules that can hold. But a complete and inclusive framework of English has not yet been created.

The first task of any natural language processing system is analysis. This is called the "parsing problem" [1] and most logic based means of NLP parse given text into a tree. Figure 1 gives an example from [1] of how the sentence "Maria Laughs" would generally be parsed.

As the sentences become more complicated, so do the associated parse trees. Additionally, there are more than a few English phrases that are ambiguous without further context. A commonly given example is "I saw the boy in the park with a telescope" [1]. In this sentence it is unclear which part is the preposition and thus, without further context, it cannot be determined whether the observer, "I", used a telescope to see the boy or whether the observer saw a boy that had a telescope. This is a major problem for processing language and, as will be expanded upon in the next chapter, most logic-based attempts at NLP either restrict their English such that ambiguous phrases are not permitted or they pre-choose interpretations for ambiguous phrases.

There have been many proposed grammars for logic based NLP. An in-depth explanation of each is outside the scope of this paper and ultimately somewhat superfluous to the proposed research; however, the considerations that should be made when constructing a grammar are important to note. Any successful form of knowledge representation, that is derived from natural

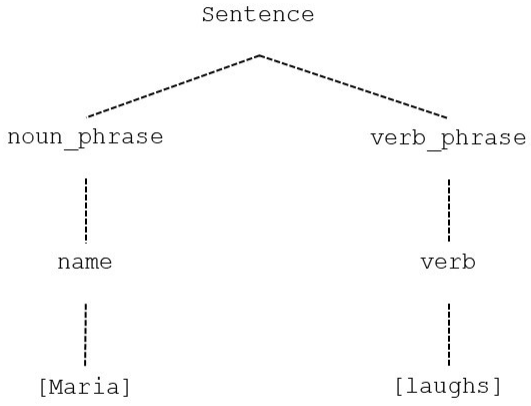


Figure 1.1: Parse Tree Recreated from [1]

language, must strike a balance between its coverage, its accuracy, and its conciseness. In this case we say that the coverage of a grammar is how much English the grammar capture, its accuracy is how well it preserves original meanings, and its conciseness is whether or not it expands or shortens representations. For a successful a few important questions to answer are: can the language capture different tenses? non-standard sentence formations? how does it deal with the prepositional problem in the previous example? The coverage of a grammar and its accuracy are closely related in that as coverage increases, it becomes harder to maintain accurate interpretations. If a grammar allows for a sentence with multiple meanings, how is accuracy maintained? Different grammars have different solutions to this problem, all of them with some sacrifice. Finally, there is the conciseness of a grammar. While it may be tempting to make a grammar that includes all possible information about a statement, this approach quickly becomes unwieldy to work with by both humans and machines. More information intensive grammars can solve the dichotomy of coverage and accuracy, at the cost of heavy computational load. These grammars are also less likely to resemble the original natural language that the grammar is attempting to represent. The open problem of finding a good means for representing natural language in less ambiguous ways is central to the proposed research and will be heavily expanded upon in the next chapter.

A use case often proposed for logic-based NLP systems is that of large scale inference. That being the ability to make inferences about some domain given a large set of predicates relating to the domain. An example of work done on this is the study from [9]. This study focuses on natural logic (see [10]) which is a subject that will be increasingly important in chapters 2 and 3.

The objective of this study was database completion, where the goal was to predict the truth of

an unseen fact and determine its inclusion in a database of true facts. The task was framed as an inference problem, involving the prediction of the query's truth based on a set of candidate premises. Traditional methods often face challenges in capturing these inferences effectively, particularly for large-scale open-domain tasks. Learning inference rules struggles with generalizing to arbitrary relations, and standard information retrieval methods may overlook subtle but semantically significant lexical differences [9].

Unlike prior work on Natural Logic that focused on inferences from a single relevant premise, this approach operates over a large set of candidate premises simultaneously. It eliminates the need for explicit alignment between a premise and the query and allows imprecise inferences with associated costs learned from data. The methodology frames inference as a unified search problem from a query to any valid supporting premise. Each transition in the search represents a reverse inference step in Natural Logic, incurring a cost reflecting the system's confidence in the validity of that step.

This approach contributes by accommodating unstructured text as the input database without assumptions about the schema or domain of the text. Additionally, it proposes the use of Natural Logic for inference without translation to formal logic syntax. In the next chapter, discussion of natural logic and natural languages will be examined more closely.

### 1.3.1 Inductive Logic Programming

One approach to NLP by way of logic programming is that of inductive logic programming. This method has been popular for some time, one of the earlier studies advocating for it premiered in 1996 and showed a clear parity or advantage to the approach when compared to statistical methods [11]. Since then much work has been done with the method as laid out in [12].

Inductive Logic Programming (ILP) stands out from many Machine Learning (ML) approaches due to its advantageous features, as highlighted in [13] and [14]. Notably, ILP exhibits data efficiency by demonstrating the ability to generalize effectively from small sets of training examples, a challenge faced by various ML techniques, including deep learning. Unlike neural systems, which may struggle with significant decreases in predictive accuracy when tested on examples with vastly different characteristics, ILP can induce hypotheses from minimal examples, sometimes even from a single instance. ILP's reliance on background knowledge (BK) represented as a logic program contributes to its unique capabilities. This representation allows ILP to learn with intricate relational information, encompassing constraints about causal networks, event calculus axioms, and theories of light. The symbolic nature of hypotheses in ILP facilitates the addition of hypotheses to BK, supporting lifelong and transfer learning seamlessly.

The expressivity of logic programs empowers ILP to learn complex relational theories encompassing diverse domains such as cellular automata, event calculus, Petri nets, answer set programs (ASP), and general algorithms. The symbolic nature further enables ILP to reason about hypotheses, leading to the acquisition of optimal programs, including those with minimal time complexity and secure access control policies. The explainability of ILP is underscored by the resemblance of logic programs to natural language, making them easily interpretable by humans. This characteristic is crucial for achieving explainable AI. For instance, [15] showcases the application of

ultra-strong ML, where a learned hypothesis not only demonstrates accuracy but also measurably enhances human performance when provided with the acquired hypothesis.

# Chapter 2

# Controlled Natural Languages

Linguists have historically been plagued by the ambiguity of English. As referenced in chapter 1 there have been attempts to fully catalog English and its rules but the problem, from a computer scientists perspective; is that it doesn't really matter what the "rules" of English are if people don't use it in accordance with the rules. In regards to natural language processing, one possible solution for this problem are controlled natural languages (CNLs). A CNL, generally, is some form of an existing natural language (English, Spanish, etc.) such that it has been restricted in some useful way. The CNLs that are discussed in this section have been restricted so that they are less ambiguous in their meanings.

## 2.1 Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language first introduced by Norbert E. Fuchs and Rolf Schwitter in 1996 [16]. One of the main goals of this language, and the goal most relevant to this research, is its goal of offering a controlled subset of English such that it was equivalent to a formal logic. The requirements for being a formal logic are somewhat strict: all statements must be unambiguous, the logic must be entirely consistent, etc. However, those strict requirements also provide a litany of benefits a particularly important benefit being that formal logics are easily machine interpretable. Therefore, ACE provides a large step in the direction of natural language processing by machines.

Beyond this, there are a few key things to know about ACE as a language. The first is the general grammar of ACE. Much of the information regarding ACE's grammar is taken from `http://attempto.ifi.uzh.ch/site/pubs/papers/ace3manual.pdf`, a manual created for the purposes of introducing people to ACE, its goals, and its grammar [17]. To understand any language, understanding its vocabulary is key. The vocabulary of ACE consists of two types of words, user defined and pre-defined. A user defined word is any word that ACE does not have inherent knowledge of and they usually take the form of nouns, verbs, adjectives, and adverbs. Most languages do not include user-defined words, so it may seem strange that ACE does, but ultimately the goal of ACE is to act as a tool for people across domains. As such, ACE has taken the approach that "content" words should be controlled and defined by the user such that they can interact with the rest of ACE's vocabulary, the pre-defined function words. Function words could also be described as the "logical" words, in that they often describe relations between objects. In English these words often take the form of determiners, conjunctions, and prepositions [17]. Note that it may seem unwieldy for ACE to expect users to manually define any and every noun, verb, or adjective they plan to use; and if that were the case it would be unwieldy. However, there are pre-existing lexica that can be

imported for a quick way to define large quantities of words.

### 2.1.1 The Grammar of ACE

Grammatically, ACE consists of two types of sentences: simple and composite. A simple sentence is, somewhat predictably, any sentence which conveys a singular event or state. Examples of simple sentences describing events and states respectively are given below.

| | |
|---|---|
| *The man buys a book.* | A present tense description of an event. |
| *The jar is empty.* | A present tense description of the state of a jar. |

Further detail regarding content in a simple sentence can be given by adding more words. For instance, the nouns *man* and *book* can be elaborated on with the use of adjectives, (*The angry man buys an old book*), by using possessives and of-prepositions, or by using proper nouns [17]. Additional information regarding the event can be added with the use of adverbs or prepositional phrases. Below is an example, taken from [17], of an information dense ACE simple sentence.

| subject | +verb | + complements | +adjuncts |
|---|---|---|---|
| *John's customer who is new* | *inserts* | *a valid card of Mary* | *manually into a slot A.* |

**example taken from [17].**

The core characteristic of a simple ACE sentence is that it has the subject+verb+complements+adjunts structure. If a sentence does not have that structure, and is still an ACE sentence, then it is a composite sentence.

Any composite sentence is built upon simple sentences through four different methods: coordination, subordination, quantification, and negation. **Coordination** is the conjoining of two simple sentences into one, usually done with *or* or *and*. A coordinated sentence will generally take the form of a sentence where the subject is shared but the predicate is split. For instance the sentence {*The man buys a book or buys a drink*} is a coordinated sentence, conjoined with *or*. The subject in the previous sentence is shared since regardless of which action he takes, the man is always the one taking said action. However, the predicate is split such that one or the other is true [17].

**Subordination** deals with relative sentences and consequential sentences. A relative sentence is one in which a clarifying sentences is nested inside of a normal simple sentence. Take the sentence *A man, who is old, buys a book*, in this case the two simple sentences *A man is old* and *A man buys a book* have been composed together. Something interesting to note is that the same meaning could be represented as a single simple using a single adjective *An old man buys a book*. This shows that composite sentences can, at times, be re-written in more simple terms without losing any form of meaning [17].

**Quantification** lets users make larger logical statements about objects. Specifically, quantification captures the logical operators "there exists" and "For all". An important thing to note is that these types of logical statements, like *There exists a shoe that every person wears*, often use the passive voice. ACE does not allow the passive voice and as such, these statements must be translated to the active voice [17].

**Negation** is, simply, the negative of some simple sentence. The negation of the example sentence from earlier would be *A man does not buy a book*. These four categories, together, make up the ways in which a sentence can be composite and when combined with the simple sentence structure give the full grammar of ACE [17].

### 2.1.2 Parsing ACE and Reasoning

ACE is parsed by the Attempto Parsing Engine (APE) which is implemented in prolog. The first part of parsing some ACE sentence is translating ACE into core ACE. In the same way that ACE is a more restrictive subset of English, core ACE is a more restrictive subset of ACE. Core ACE eliminates all of the equivalent syntaxes that exist within ACE. In this way, any meaning in core ACE has a **unique** way of being represented which massively cuts down on the computational complexity of working with ACE. Once a statement exists as core ACE the parsing engine can translate the statement into DRS (Discourse Representation Structures), a separate first-order logic. While ACE does have its own reasoning engine, called RACE (The Attempto Reasoner), this reasoner is limited in the scope of its abilities [18]. Therefore, if the goals is any complex reasoning, it becomes necessary to acquire ACE statements as more widely-used formal logics such as DRS. It is important to remember that ACE is translatable to DRS *and vice-versa*. Thus if the need arises for a user to present some formal logic statements in natural language, ACE can serve as a quick and easy method of obtaining that natural language representation. In chapter 4 there will be further discussion on the ways in which ACE translates natural English to its controlled version. For that discussion it is important to understand that when English is being translated into ACE, the interpreter will return the suggested translations and let the user decide whether the interpretation is valid. This step seems ripe for integration with statistical methods.

### 2.1.3 ACE as Knowledge Representation

ACE's attributes make it particularly good at knowledge representation because it is both machine and human interpretable. Here are key reasons why this dual interpretability is significant. First, ACE enables collaboration between humans and machines in knowledge-intensive tasks. People excel at expressing domain knowledge in natural language, while machines are good at processing and reasoning with formal representations. The interpretability of ACE by both allows for collaboration and communication in tasks such as ontology development and knowledge engineering [19].

Next, Many domain experts may not have expertise in formal languages or logic. ACE's interpretability by humans allows these experts to contribute to the development of knowledge bases and ontologies without requiring specialized training in formal methods. This accessibility is crucial for democratizing the creation and maintenance of knowledge representations. Furthermore, humans find it more natural and intuitive to express concepts and relationships in their domain using natural language. ACE provides a human-friendly input method, making it easier for users to articulate their knowledge without the need for mastering complex formal languages [19].

Finally, ACE's interpretability by machines allows for the validation and correction of expressions. Users can receive feedback from both automated tools (e.g., ACE Editor), ensuring that

the expressions adhere to the controlled vocabulary and grammar rules. This validation process improves the quality of knowledge representations.

## 2.2 Controlled English to Logic Translation

Controlled English to Logic Translation (CELT) is a system for translating English to logic built upon the foundations laid by out Attempto Controlled English (ACE). ACE, focused on the design of a controlled subset of English, CELT on the other hand specifies a larger system for translating controlled English directly into logic. CELT, like ACE, uses a simplified structure. However, in contrast to ACE, CELT does not limit the different meanings that words can have. CELT does this by relying on SUMO and Wordnet, large databases of ontological and lexicographic meanings respectively. Recall that ACE left it up to users to define most nouns, adjectives, etc, CELT on the other hand has been pre-integrated with a wide range of meaning such that it is useful out of the box [20].

When the goal is parsing, users will enter queries in a controlled English and then specify a grammatical format. The controlled grammar used by CELT is extensive but, like all CNLs, is somewhat constrained so that interpretations remain unique. This deals with any potential issues regarding the ambiguity of a query. Some approaches to NLP, generally statistical ones, rely on giving the model a light understanding of natural language and responding from there. The power of CELT comes from the fact that any query given will be entirely understood by CELT immediately thus reasoning can be done reliably [20].

When parsing CELT begins by, determining parts of speech and creating a parse tree. It uses the word order and word senses provided by WordNet, augmented with common proper names, to identify parts of speech. After determining word senses, CELT translates them to SUMO terms. The logical form generated is suitable for deductive inference using a standard theorem prover, allowing for deductive reasoning. This contrasts with statistical language understanding approaches that yield non-logical results. While WordNet provides valuable information, it lacks the logical consistency and deductive support of SUMO, which can deduce intricate relationships and conclusions based on formally specified definitions. This distinction emphasizes the complementary roles of WordNet and SUMO in CELT's processing.

CELT's research aims to enhance the comprehensibility of a wider range of sentences while ensuring a thorough understanding of sentence semantics. The controlled English syntax of CELT provides lexical semantics, meaning that sentence meaning is solely determined by its structure rather than the broader context (e.g., paragraph, document, conversation). This transformation of English syntax into a formal language allows for the division of application tasks into machine-handled components (e.g., extracting sentence sense) and human-handled components (e.g., interpreting extensive text or conducting semantically rich Internet searches). The conversion from natural language to controlled English aligns the resulting fragment with the formal language (formal logic) used for inferencing with extracted text. Thus, CELT's goal surpasses that of ACE's, and although CELT has limitations, its completeness improves with the expansion of known meanings in SUMO WordNet, making it a more comprehensive method for translating natural language to logic over time [20].

### 2.2.1 SUMO

SUMO was developed as a formal ontology that could be utilized across different domains and applications. It represents a merger of several existing ontologies and is designed to be highly expressive, capturing a wide range of concepts and their interrelations. SUMO includes a large number of classes [21], relations, and axioms that can be used as a common ground for knowledge representation in diverse areas [22].

The connection between CELT and SUMO is notable in the context of knowledge engineering and ontology development. CELT's automated translation of controlled English sentences into formal logic facilitates the integration of knowledge expressed in controlled English with formal ontologies like SUMO. This interoperability is crucial for ensuring that the representations created using CELT align seamlessly with established ontological frameworks, enabling a more standardized and widely applicable approach to knowledge representation.

SUMO is heavily connected with the semantic web, a vision of the future of the web introduced by Tim-Berners Lee in [23]. A more in-depth explanation of the semantic web will be provided in Chapter 3, in the mean-time the semantic web can be described as the goal of a web which is understandable by both machines and humans. One approach to accomplishing this goal is using a "universal" language such that content can be understood equally by virtual and real users. SUMO provides a collective well of meaning that both sides can draw upon as a shared means of interpreting content [24]. Chapter 3 will go on to discuss the importance of CNLs and collected ontologies in regards to the future of the semantic web.

# Chapter 3

# The Semantic Web

There are many different reasons to design and use a CNL. A few of the more popular reasons include One of the most popular uses for CNLs is integration with the semantic web. This chapter will provide a brief introduction to the semantic web, its uses, and its relation to CNLs and ontologies at large.

## 3.1 Introduction to The Semantic Web

The semantic web is an idea introduced by Tim Berners-Lee, James Hendler and Ora Lassila in their 2001 article ”A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities” [23]. As opposed to the traditional vision of the web, the semantic web imagines a future where web pages are not just human readable; but also machine readable. This may seem strange, since much of the web is already machine readable (think html, xml, etc.) but that is the ”structure” of the web, not the actual content. Only recently, have we been able to ask a virtual agents to understand the actual content of the web. The semantic web has always been closely related to the field of knowledge representation. Indeed, the original paper introducing the idea has a whole section titled ”Knowledge Representation” [23]. The task of finding a universal way of representing knowledge, such that it is interpret able by humans and machines, is core to the idea of the semantic web.

The Semantic Web is characterized by its distributed and heterogeneous nature, representing a significant advancement in the evolution of the World Wide Web (WWW). It embodies two key developmental visions for the future of the Web: the enhancement of usability as a collaborative medium and the facilitation of machine understanding of its contents [2]. Tim Berners-Lee, credited with inventing the WWW and contributing to the Semantic Web, asserts that the latter is an extension of the current Web, endowing information with well-defined meanings to enhance collaboration between computers and humans [23].

Distinguishing itself through a more meaningful representation of information for both humans and computers, the Semantic Web employs machine-readable descriptions of its contents and services. This approach allows for the automatic annotation, discovery, publication, advertisement, and composition of services, thus promoting interoperability and knowledge sharing across the Web. As previously stated, the primary objective is to render information on the Web accessible and comprehensible to both humans and machines.

Both the Semantic Web and Web services are conceptualized as sets of resources identified by Uniform Resource Identifiers (URIs). While Web services utilize HTTP to display page contents, the Semantic Web strives for machine readability through the semantic representation of data in

resources. Various tools and applications leveraging Semantic Web technologies have recently become available.

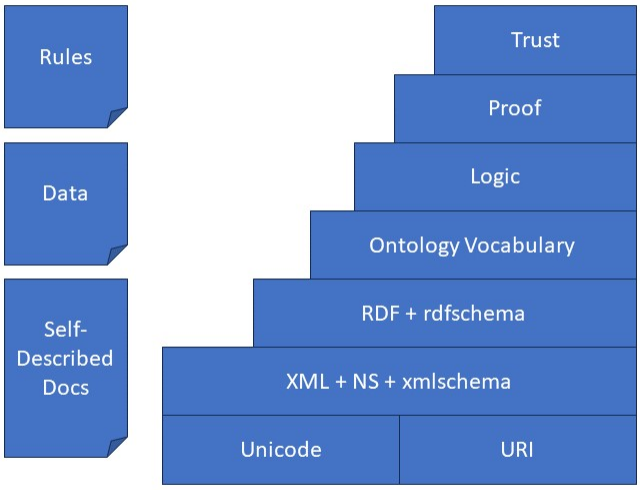


Figure 3.1: Semantic Web Layers Recreated from [2]

The architecture of the Semantic Web, as depicted in Figure 3.1 (recreated from figure 1 of [2]), consists of several layers, briefly outlined as follows:

- URI and Unicode: Utilizing a uniform system of identifiers (URIs) to identify and locate resources on the Web, the foundation of the Web is established. Unicode serves as the standard for computer character representation.
- Extensible Markup Language (XML): As a machine-readable markup language, XML is recognized for its flexible text format. It plays a crucial role in describing data on the Web and is foundational for a growing number of software development activities.
- Resource Description Framework (RDF): Serving as the first layer of the Semantic Web, RDF is a framework for representing metadata and describing the semantics of information about Web resources in a machine-accessible manner. It uses URIs to identify and describe relations between resources in a graph model.
- Ontology Vocabulary: This language provides a common vocabulary and grammar for published data, offering a semantic description of the data to preserve ontologies and enable communication between different parties.

- Logic and Proof: In the Semantic Web, systems are built following a logical approach that considers the structure of ontologies. Reasoners are employed to check and resolve consistency problems and redundancies, with a reasoning system facilitating new inferences.
- Trust: Serving as the final layer of the Semantic Web, the Trust component addresses the trustworthiness of information on the Web, providing assurance of its quality.

### 3.1.1 Ontologies

As discussed in section 2.2.1, ontologies are core to the creation of the semantic web. An ontology encompasses four principal components: concepts, instances, relations, and axioms [2].

- Concept (or class/term): Serving as an abstract group, set, or collection of objects, a concept is a foundational element within the domain. Typically, it denotes a group or class whose members share common properties. Its representation in hierarchical graphs mirrors object-oriented systems, featuring a ”super-class” signifying the higher or ”parent class” and a ”subclass” denoting the subordinate or ”child class.” For example, a university may be portrayed as a class with various sub classes such as faculties, libraries, and employees.
- Instance (or individual): Positioned at the ground level of an ontology, an instance represents a specific object or element within a concept or class. For instance, ”Jordan” could function as an instance of the class ”Arab countries” or, more generally, ”countries.”
- Relation (or slot): Employed to articulate relationships between two concepts in a given domain, a relation elucidates the connection between the first concept (in the domain) and the second concept (in the range). For instance, the relationship ”study” could be delineated between the concept ”person” (in the domain) and ”university” or ”college” (in the range).
- Axiom: Utilized to impose constraints on the values of classes or instances, axioms are typically expressed using logic-based languages like first-order logic. They serve to validate the consistency of the ontology by articulating restrictions on classes or instances.

Over time, ontology has gained prominence as a focal point of research across various disciplines, aiming to enhance comprehension and establish consensus within specific knowledge domains. Its utility extends to fostering knowledge sharing among systems and individuals. Originating in AI laboratories, ontology has transcended its initial application, finding relevance in diverse fields such as the Semantic Web, Semantic Web Service Discovery, Artificial Intelligence (AI), Multi-agent systems, Search Engines, E-Commerce, and Interoperability [2].

In the context of the Semantic Web, ontology assumes a pivotal role by supporting information exchange in distributed environments, contributing to the machine-processable representation of data. This representation is considered an extension of the existing Web. In Semantic Web Service Discovery within the e-business domain, ontology aids in identifying optimal matches for merchandise seekers and facilitates responses for online travel customers [2].

Within AI, ontology development focuses on fostering knowledge sharing, reuse, and processing efficiency across domains for programs, services, agents, or organizations. In Multi-agent

systems, ontology ensures a shared understanding of domain knowledge, promoting seamless communication and reducing misunderstandings among agents. Search Engines utilize ontology, often in the form of thesauri (large collections of meanings), to identify synonyms for search terms, enhancing internet searching efficiency [2].

In E-Commerce, ontology facilitates communication between sellers and buyers by describing merchandise and enabling machine-based communication. The challenge of interoperability, involving the integration of heterogeneous and distributed systems, is addressed through ontology, which serves to integrate diverse application systems [2].

In the realm of services, ontology assumes a central role by providing detailed descriptions of services, terms, and their relationships within the application domain. This facilitates the explicit representation of knowledge within the domain and supports the inference of implied knowledge through declared descriptions [2].

An illustrative example is provided to underscore ontology's significance as the backbone of the Semantic Web. In this scenario, ontology is employed to match services with semantic meanings. When a service requester invokes a service (e.g., buying a car), the service request information is annotated in metadata. Service providers also describe and advertise their services in metadata. The service match engine, using ontology, identifies relevant terms such as "automobile" and "vehicle" with "car," enabling the engine to infer whether the service request has been satisfied [2].

### 3.1.2 Why the Semantic Web?

There are many visions of the future of the web and so the question must be asked; why choose this one over others? The core argument in favor of the semantic web is that it would greatly benefit the ability for people to interface with the web. The web is a vast expanse of information, so vast, that it can be difficult for people to draw any meaningful conclusions from the information. It may be advantageous to have a large collection of information, however the real goal of collecting any amount of information is to draw some knowledge from it. This requires some level of synthesis, the collecting of data into a conclusion, that can be difficult for people to do. This is especially true when the information that must be analyzed is exorbitantly large. Thankfully, there is a solution. Machines are very good at collecting and analyzing large sets of data, and if they could understand the meaning of such data then it may be a simple task for them to synthesize some conclusions from said data. Thus we have a goal. Allow machines to understand the content of the data that they are traversing, such that the can synthesize meaning from it. Notice that this perfectly aligns with the goal of the semantic web, to allow content to be machine understandable. In this revelation, the value of the semantic web becomes clear.

This benefit has become even more prescient in the face of large language models. LLMs are great at understanding content and synthesizing conclusions, however they are built on shaky foundations. Since there is currently no "correct" interpretation of content found on the web, any conclusions produced by LLMs are at best, informed guesses. If however, a model had access to a deterministic translation of the content it is analyzing; the task of analysis and synthesis of conclusions becomes drastically simpler.

### 3.1.3 Growing the Semantic Web

The semantic web has now been an idea for over two decades. In that time, much progress has been made but the question of ”Where do we go next?” is one which remains open. This section will present a few areas of development that the semantic web is or could be used for. Much of this sections information will be drawn from a paper by Archana Patel and Sarika Jain titled ”Present and future of semantic web technologies: a research statement” [25].

The Semantic Web’s data space is characterized by its distributed, dynamic, and privacy-sensitive nature. To foster its growth, three domains play crucial roles: Computational Intelligence (CI), Evolutionary and Swarm Computing, and Knowledge Representation Methods. Swarm Computing handles large-scale data storage and reasoning, CI addresses vagueness and uncertainty, and Knowledge Representation ensures consistent and coherent data storage.

In the realm of Computational Intelligence (CI), methods such as Evolutionary Computation, Fuzzy Logic, and Artificial Neural Networks (ANNs) are employed to tackle the Semantic Web’s challenges, including vastness, uncertainty, and inconsistency. CI techniques are strategically applied to issues related to Vastness and Tractability, Vagueness and Uncertainty, and Divergence and Inconsistence. Supervised ANNs are extensively used for Ontology Alignment, addressing challenges related to structured data processing and ontology learning.

For Ontology Alignment, Recursive Neural Network (RNN) models are employed to process structured data accurately seen in [26] and [27]. The Interactive Activation and Competition neural network fetches global optimal solutions, resolving constraint satisfaction issues. Ontology Learning, involving the semi-automatic or automatic creation of ontologies, utilizes unsupervised ANNs such as Self-Organization Map (SOM) for ontology construction and enrichment. Projective Adaptive Resonance Theory Neural Network contributes to automatic ontology construction from the web.

In Evolutionary and Swarm Computing, algorithms address optimization problems with large and dynamic search spaces. These approaches are vital for scalable, adaptive, and robust handling of large-scale Semantic Web data. Swarm-based techniques, rooted in collective behavior, enable reasoning in entirely decentralized and self-organized storage systems. However, they trade deterministic guarantees for benefits, necessitating a balance in distributed systems [25].

Knowledge Representation of Smarter Data emphasizes the importance of representation schemes, introducing Extended Hierarchical Censored Production Rules (EHCPR). EHCPR effectively visualizes real-world entities and manages large data sets. The structure employs defining and characteristic properties, contributing to effective knowledge representation. Additionally, efficient query languages are crucial for retrieving relevant output from stored knowledge. EHCPR emerges as an integrated approach fulfilling the requirements for developing intelligent applications, providing a structured representation for the Semantic Web.

## 3.2 CNLs and The Semantic Web

As should be clear from the CNL and semantic web sections. The goal when designing a CNL and the goals of the semantic web are very much aligned. There have already been proposals for the

use of CNLs with the semantic web, this will be the focus of the following section.

The semantic web has many different layers, as explained in the previous sections. The layer which has the most to gain from CNLs is called the ontology inference layer. The conventional web is primarily designed for human consumption, posing a challenge for machines to comprehend its content effectively. Current technologies, including web browsers, servers, and search engines, lack the capability to distinguish diverse types of information, hindering the web's functionality. Machines are confined to transmitting and presenting information without the ability to assist in processing it. The semantic web aims to address this limitation by creating a web environment where both humans and machines can understand the information available online. Achieving this goal requires representing information in a manner that allows machines to access its meaning, commonly referred to as "semantics." The Ontology Inference Layer serves as a specific representation designed to provide machine-accessible semantics for information on the web [3]. See figure 3.2 recreated from [3] for an example of an ontology interface layer ontology.

```
class-def Product
slot-def Price
  domain Product
slot-def ManufacturedBy
  domain Product
class-def
PrintingAndDigitalImagingProduct
  subclass-of Product
class-def HPProduct
  subclass-of Product
  slot-constraint ManufacturedBy
    has-value "Hewlett Packard"
class-def Printer
  subclass-of
PrintingAndDigitalImagingProduct
slot-def PrinterTechnology
  domain Printer
slot-def Printing Speed
  domain Printer
slot-def PrintingResolution
  domain Printer
class-def PrinterForPersonalUse
  subclass-of Printer
class-def HPPrinter
  subclass-of HPProduct and Printer
class-def LaserJetPrinter
  subclass-of Printer
  slot-constraint PrintingTechnology
    has-value "Laser Jet"
...
```

Figure 3.2: Ontology example recreated from [3]

There have been many CNLs created for the semantic web. The survey done in [28] provides a good look at some of the most prominent.

PENG, a CNL, is designed for crafting unambiguous and precise specifications through a restricted grammar and a domain-specific lexicon. It emphasizes data efficiency by enabling hypothesis induction from small examples. ClearTalk (CT) serves as a knowledge formulation language with a balanced degree of formality and expressiveness, allowing for automatic translation to a knowledge representation structure. PENG-D extends PENG, incorporating support for ontology construction and employing Description Logic Programs (DLP) for enhanced expressivity.

Sydney OWL Syntax (SOS) emerges as a response to limitations in prior CNLs, seeking compatibility with OWL 1.1 for ontology expression.OWL Simplified English adopts a finite state

language approach for ontology editing, recognizing distinct Part Of Speech (POS) tags to capture ontology operations while aiming to reduce structural ambiguity. Semantic Query and Update High-Level Language (SQUALL) simplifies semantic querying and updates on RDF graphs by combining expressiveness close to SPARQL 1.1 with a natural syntax based on Montague grammars. AIDA proposes a CNL for extending nanopublications, enforcing sentences to be Atomic, Independent, Declarative, and Absolute, introducing the nanobrowser portal for tracking scientific results with URI-assigned sentences.

When comparing these CNLs some have clear advantages over the others. For instance [29] was a study attempting to rank CNLs by understand ability. The study demonstrated that ACE, was significantly better understood compared to MLL, a simplified version of the Manchester OWL Syntax, as indicated by participant evaluations. ACE not only required less time for learning but was also perceived as more understandable. These findings suggested that CNLs, particularly ACE, should be preferred over traditional ontology languages like Manchester OWL Syntax, especially in scenarios involving users with minimal training or unfamiliarity with formal notations.

Certain patterns, notably plain subtype statements, exhibited a particularly strong positive effect of controlled natural language over classical ontology languages. However, the study acknowledged the need for further exploration into the learning curves of both language types over extended periods, considering the possibility of common logic notations catching up or surpassing CNL in understandability.

While controlled natural languages presented minor challenges, such as increased difficulty in computer parsing and potential confusion with full natural language, the study emphasized their potential advantages in user interfaces. The results suggested that interfaces based on CNLs, like ACE, had the potential to be more user-friendly and faster to comprehend than those relying on conventional ontology languages. The study underscored the importance of past efforts in designing CNLs for the Semantic Web and encouraged continued initiatives in this direction.

# References


[1] Veronica Dahl. Natural language processing and logic programming. *The Journal of Logic Programming*, 19-20:681–714, 1994. Special Issue: Ten Years of Logic Programming.

[2] Mohammad Mustafa Taye. Understanding semantic web and ontologies: Theory and applications, 2010.

[3] O. Lassila, F. van Harmelen, I. Horrocks, J. Hendler, and D.L. McGuinness. The semantic web and its languages. *IEEE Intelligent Systems and their Applications*, 15(6):67–73, 2000.

[4] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*, pages 1–10. Cambridge University Press, 2014.

[5] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*, pages 11–39. Cambridge University Press, 2014.

[6] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*, pages 152–191. Cambridge University Press, 2014.

[7] Michael Gelfond and Jorge Lobo. Authorization and obligation policies in dynamic systems. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, pages 22–36, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*, pages 61–85. Cambridge University Press, 2014.

[9] Gabor Angeli and Christopher D. Manning. NaturalLI: Natural logic inference for common sense reasoning. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 534–545, Doha, Qatar, October 2014. Association for Computational Linguistics.

[10] Bill MacCartney and Christopher D. Manning. Natural logic for textual inference. In Satoshi Sekine, Kentaro Inui, Ido Dagan, Bill Dolan, Danilo Giampiccolo, and Bernardo Magnini,

editors, *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200, Prague, June 2007. Association for Computational Linguistics.

[11] Raymond J. Mooney. Inductive logic programming for natural language processing. In Stephen Muggleton, editor, *Inductive Logic Programming: Selected papers from the 6th International Workshop*, pages 3–22. Springer Verlag, Berlin, 1996.

[12] Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen Muggleton. Inductive logic programming at 30. *Machine Learning*, 111:1–26, 01 2022.

[13] Andrew Cropper. Forgetting to learn logic programs. *ArXiv*, abs/1911.06643, 2019.

[14] Andrew Cropper. Playgol: learning programs through play, 04 2019.

[15] Stephen Muggleton, Wang-Zhou Dai, Claude Sammut, Alireza Tamaddoni-Nezhad, Jing Wen, and Zhi-Hua Zhou. Meta-interpretive learning from noisy images. *Machine Learning*, 107:1–22, 07 2018.

[16] Norbert Fuchs and Rolf Schwitter. Attempto controlled english (ace). *CoRR*, cmp-lg/9603003, 03 1996.

[17] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto controlled english (ace)language manualversion 3.0. Technical report, 1999.

[18] Norbert Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. volume 2006, pages 664–669, 05 2006.

[19] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. *Attempto Controlled English for Knowledge Representation*, pages 104–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[20] Adam Pease and John Li. Controlled english to logic translation. 2010.

[21] Adam Pease and Christoph Benzmüller. Ontology archaeology: Mining a decade of effort on the suggested upper merged ontology. 09 2010.

[22] Ian Niles and Adam Pease. Towards a standard upper ontology. pages 2–9, 10 2001.

[23] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*, 05 2001.

[24] Adam Pease, Ian Niles, and John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. 01 2002.

[25] Archana Patel and Sarika Jain. Present and future of semantic web technologies: a research statement. *International Journal of Computers and Applications*, 43:1–10, 01 2019.

[26] Samuel R. Bowman. Can recursive neural tensor networks learn logical reasoning?, 2014.

[27] Samuel R. Bowman, Christopher Potts, and Christopher D. Manning. Recursive neural networks for learning logical semantics. *CoRR*, abs/1406.1827, 2014.

[28] Hazem Safwat and Brian Davis. Cnls for the semantic web: a state of the art. *Language Resources and Evaluation*, 51, 03 2017.

[29] Tobias Kuhn. The understandability of owl statements in controlled english. *Semantic Web*, 4:101–115, 01 2013.