EXPLORING SEMANTIC HIERARCHIES TO IMPROVE RESOLUTION THEOREM PROVING ON ONTOLOGIES

by

Stanley C. Small

A Thesis Submitted in Partial Fulfillment
of the Requirements for a Degree with Honors
(Computer Science)

The Honors College
University of Maine
May 2019

Advisory Committee:
- Dr. Torsten Hahmann, Assistant Professor[1], Advisor
- Dr. Mark Brewer, Professor of Political Science
- Dr. Max Egenhofer, Professor[1]
- Dr. Sepideh Ghanavati, Assistant Professor[1]
- Dr. Roy Turner, Associate Professor[1]

[1]School of Computing and Information Science

ABSTRACT

A resolution-theorem-prover (RTP) evaluates the validity (truthfulness) of conjectures against a set of axioms in a knowledge base. When given a conjecture, an RTP attempts to resolve the negated conjecture with axioms from the knowledge base until the prover finds a contradiction. If the RTP finds a contradiction between the axioms and a negated conjecture, the conjecture is proven.

The order in which the axioms within the knowledge-base are evaluated significantly impacts the runtime of the program, as the search-space increases exponentially with the number of axioms.

Ontologies, knowledge bases with semantic (and predominantly hierarchical) structures, describe objects and their relationships to other objects. For example, a 'Sedan' class might exist in a sample ontology with 'Automobile' as a parent class and 'Minivan' as a sibling class. Currently, hierarchical structures within an ontology are not taken into account when evaluating the relevance of each axiom. Instead, each predicate is automatically assigned a weight based on a heuristic measure (such as the number of terms or the frequency of predicates relevant to the conjecture) and axioms with higher weights are evaluated first. My research aims to intelligently select relevant axioms within a knowledge-base given a structured relationship between predicates. I have used semantic hierarchies passed to a weighting function to assign weights to each predicate. The research aims to design heuristics based upon the semantics of the predicates, rather than solely the syntax of the statements.

I developed weighting functions based upon various parameters relevant to the ontological structure of predicates contained in the ontology, such as the size and depth of a hierarchy based upon the structure of the ontology. The functions I have designed calculate weights for each predicate and thus each axiom in attempts to select relevant axioms when proving a theorem. I have conducted an experimental study to determine if my methods show any improvements over current reasoning methods. Results for the experiments conducted show promising results for generating weights based on semantic hierarchies and encourage further research.

ACKNOWLEDGEMENTS

Many thanks are given to Dr. Hahmann. This work could not be completed without his continued support and encouragement. Despite his tremendously busy schedule, he always made time to meet and answer questions.

Robert Powell also proved instrumental to the process. He wrote a utility which converts Common Logic Interchange Format (CLIF) into web ontology language (OWL). His work streamlined the testing process and allowed me to find necessary results.

The thesis committee was also instrumental in producing an undergraduate thesis of scale.

List of Figures

List of Tables

1 INTRODUCTION

The rules of logic enable one to prove theorems from axioms stored in a knowledge base. Axioms, asserted facts typically expressed in a formal manner, provide a computer program with tools to confirm or refute conjectures without additional user input. Because computers excel at simple and repetitive tasks, one can witness the applications of automated theorem proving in fields which rely heavily on "knowledge acquisition and information retrieval" [**sanchez2012ontology**]. The ability for machines to deduce logically valid conclusions has applications in artificial intelligence and a variety of scientific domains [**urban2011overview**]. Automated theorem proving provides a versatile method for reasoning with a set of facts, and has been used to prove and verify proofs of multiple theorems. The four color map theorem, obtained by an automated proof in 1976 by a general-purpose theorem-proving software in 2005 remains a notable example [**gonthier2008formal**]. Moreover, advances have been made in work on the Kepler conjecture and in finding optimal solutions for a Rubik's Cube. The general-purpose nature of automated theorem proving yields applications to a variety of problems. However, automated theorem proving programs often neglect semantic knowledge embedded in an ontology.

Ontologies provide a "common vocabulary" for researchers to speak about a specific domain by describing entities and the relationships between them [**noy2001ontology**]. A formal description of a specific environment provides researchers and machines with a shared understanding by aiming to capture the semantics of a domain's concepts and relations. Some relationships between an ontology's terms may be explicitly defined, but many are implicit. For example, three axioms one might find in a knowledge base are below.

$$SubaruLegacy(myCar)$$
$$\forall x\ SubaruLegacy(x) \rightarrow Sedan(x)$$
$$\forall x\ Sedan(x) \rightarrow Automobile(x)$$

The first axiom asserts my car is a Subaru Legacy. The next states a Subaru Legacy is a sedan.

The last asserts all sedans are automobiles. While one can easily deduce the fact my car is an automobile, no single axiom explicitly describes such a statement. Fortunately, formal logic defines rules of inference which allow one to transform established facts into new conclusions solely based on the syntax of these statements. Both humans and computers can clearly distinguish well formed statements ($x + y = 4$) from those which are not ($x4y+ =$). Beyond the syntax of statements, ontologies and logics also define the semantics or meaning of sentences (i.e. declaring $x + y = 4$ is true when $x = 1$ and $y = 3$ but false when $x = 0$ and $y = 1$). Thus, the sentence $\exists x, y\ (x + y = 4)$ asserting that $x + y = 4$ is true for some numbers is true, whereas $\forall x, y\ (x + y = 4)$, asserting that $x + y = 4$ is true for all possible combinations of x and y is false.

Like many taxonomies, or schemes of classification, one can often form maps of relationships within an ontology which resemble a hierarchy. Knowledge encoded in semantic hierarchies could help an automated theorem prover determine which axioms might be most helpful when attempting to prove a specific conjecture. This work attempts to improve automated theorem proving with ontologies by identifying relevant facts, and ignoring those less likely to yield a proof.

2 BACKGROUND AND RELATED WORK

2.1 Ontologies

The word ontology (”study of being”) combines Greek *onto-* (”being”) and *-logia* (”logical discourse”). The act of studying knowledge and existence in philosophy has given birth to the study of formal logic and automated reasoning in computer science. Researchers often use ontologies to share information among people or computer programs, to enable domain knowledge reuse, to make definitions of a particular domain explicit, or to analyze domain knowledge [**noy2001ontology**].

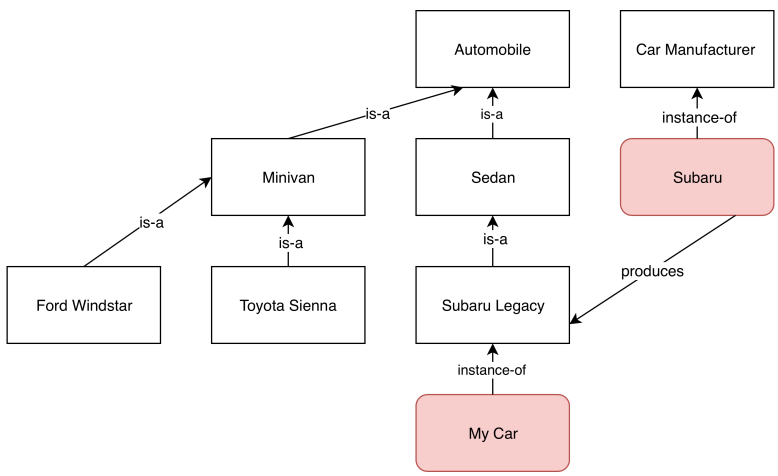

Figure 1: This sample ontology, with inspiration from an example by Natalya F. Noy, describes entities and relations in the ’automobiles’ domain. The ontology serves to provide a ”formal explicit description” of classes (outlined in black) along with properties which describe relationships between classes (such as how Subaru Legacy is produced by Subaru). While not displayed in the figure, an ontology also defines property restrictions within the domain (so an instance of the Subaru class cannot produce a car manufacturer). The ontology, along with individual instances of classes (highlighted in red) constitutes a knowledge base [**noy2001ontology**].

In reality, few differences between an ontology and a knowledge base exists. Knowledge engineers must traverse a ”fine line where the ontology ends and the knowledge base begins” [**noy2001ontology**]. At the least, an ontology defines categories (or classes) and relationships among objects. One can think of an ontology as a ”vocabulary” used to describe a domain [**russell2016artificial**]. Typically, both classes and relationships between classes can be arranged as hierarchies (see Figure **??**), which are here referred to as semantic hierarchies. When designing an ontology, one must decide the scope and organization of the knowledge, along with the language used.

2.1.1 Class Inheritance and Semantic Reasoning

Many relations within an ontology serve to organize classes. For example, in the 'automobiles' ontology, most of the classes are organized by "is-a" relationships and classes inherit attributes such as domain and range restrictions. For example, my car would inherit properties of the 'Automobile', 'Sedan', and 'Subaru Legacy' classes, such as having four seats.

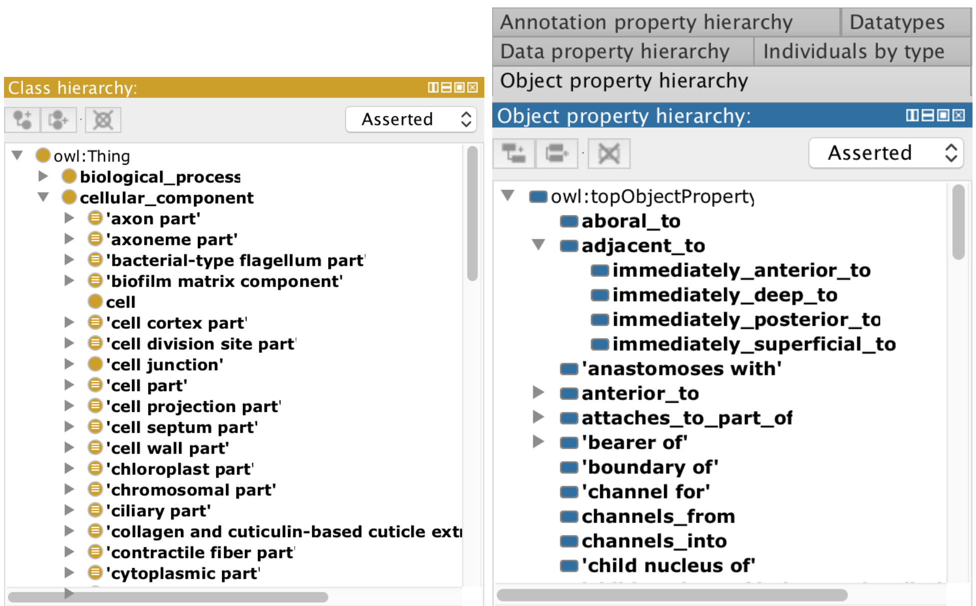

Figure 2: Hierarchies describing classes and relationships within the commonly used Gene Ontology (not used for experiments) can be viewed in Protégé, with inferred hierarchies generated by Pellet reasoner [**gennari2003evolution**].

Software tools referred to as semantic reasoners, or simply reasoners, can infer logical consequences from a set of axioms. Because an ontology may not explicitly define all relationships between classes and properties, one may use a reasoner to deduce implicit knowledge. A reasoner can generate a more complete view of an ontology, specifically complete hierarchies describing classes and relationships. These hierarchies are referred to as semantic hierarchies.

Ontologies, especially those used in research, can contain hundreds or thousands of classes and relationships but only a small fraction of those are likely needed for any specific proof. By consulting knowledge embedded in the semantic hierarchies for a specific ontology one could possibly reduce the time needed to prove a specific conjecture when many irrelevant axioms exist.

2.1.2 First-order Logic Ontologies

Automatic theorem proving requires a logic defining the syntax of valid statements to run without additional user input. Formal logics like first-order logic, also known as predicate logic and first-order predicate calculus, define a structure for statements which can be used to form logical and mathematical proofs. Consider the following set of asserted facts expressed in first-order predicate logic, commonly referred to as axioms.

$$isSedan(myCar)$$
$$\forall x \; isSedan(x) \rightarrow hasFourSeats(x)$$

The first axiom asserts my car is a sedan. The second axiom asserts all sedans have four seats. By expressing facts in a formal notation, one makes proofs using such statements mechanical and easily parsed by a computer. Ontologies used for experiments are described used Common Logic, a formal logic based on first-order logic. Predicates describe objects in a knowledge base. In the cases above, $isSedan()$ would serve as a predicate acting on a $myCar$ object in the former, and a variable labeled x in the latter. Variables in first-order logic are quantified, meaning the application of a variable is defined for either some ($\exists$) or all ($\forall$) objects in the domain.

2.2 Theorem Proving

Automated theorem proving depends on having an established logic for expressing facts (such as Common Logic), a method of generating new facts without requiring additional knowledge, and a strategy for searching through all possible new facts one could generate to reach a specific goal (such as proving a conjecture).

2.2.1 Inference Rules

One can use axioms to derive facts which logically follow using inference rules. The two previous statements do not directly state the my car has four seats. However, one can derive the statement

$hasFourSeats(myCar)$ by using the inference rule *modus ponens*, defined below.

$$\begin{array}{c} A \\ A \rightarrow B \\ \hline \therefore B \end{array} \tag{1}$$

One can think of A and B as variables representing statements, and any statements can replace them. In the example above, one can replace A with $isSedan(myCar)$ and B with $hasFourSeats(myCar)$ after instantiating x with $myCar$ (which is possible because x is bound by the universal quantifier $\forall$ and we can replace x with anything defined in the domain). Therefore, one can assert $hasFourSeats(myCar)$ is true, without the statement having been defined explicitly as an axiom. An inference rule defines a valid rule for statements and always generates true statements when the assumed premises are true.

2.2.2 Resolution

Automated theorem proving requires a set of axioms and a set of rules to generate new facts, but also a strategy to search through the possible applications of the inference rules. Knowledge bases can grow quite large, and generating all possible facts based on a given set of axioms often remains impractical or unfeasible. Resolution exists as historically significant and widely used method for automated theorem proving [**ertel2018introduction**].

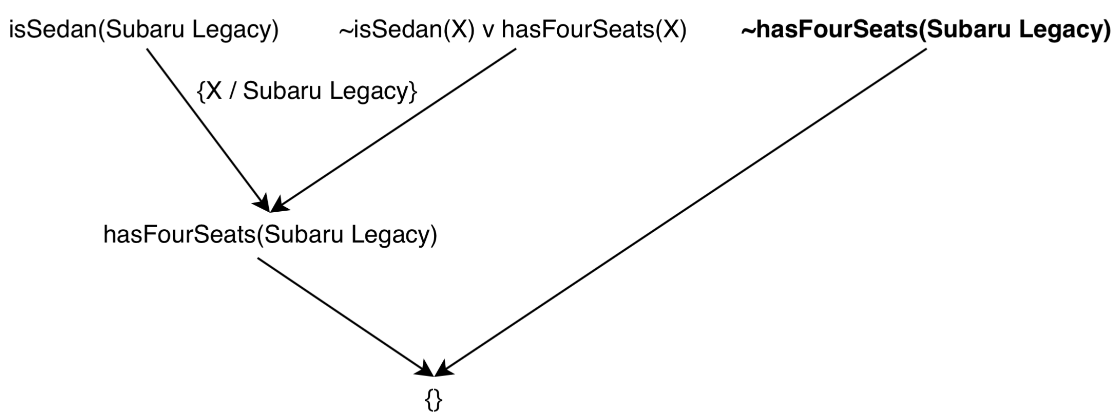

Figure 3: The figure above displays a resolution tree for the inference rule described in the previous section. The bold statement shows the negated conjecture. The tree also displays x bound to $SubaruLegacy$.

In order to use resolution as a proof technique, axioms must first be expressed in Conjunctive Normal Form (CNF), also known as clausal form. One may follow a 7-step procedure of converting the set of facts into a conjunction of disjunctions. The process eliminates biconditionals, implications, and quantifiers so the second axiom $\forall x\ isSedan(x) \rightarrow hasFourSeats(x)$ becomes $\neg isSedan(x) \vee hasFourSeats(x)$. One can then resolve the statements by instantiating the variable x with *SubaruLegacy* in a process called unification, binding the variable.

$$\frac{isSedan(myCar), \neg isSedan(x) \vee hasFourSeats(x)}{hasFourSeats(myCar)}$$

Finally, one can resolve the axiom with the negated conjecture, proving the statement true.

```
1 (all x (isSedan(x) -> hasFourSeats(x))) # label(non_clause).  [assumption].
2 hasFourSeats(myCar) # label(non_clause) # label(goal).  [goal].
3 -isSedan(x) | hasFourSeats(x).  [clausify(1)].
4 isSedan(myCar).  [assumption].
5 hasFourSeats(myCar).  [resolve(3,a,4,a)].
6 -hasFourSeats(myCar).  [deny(2)].
7 $F.  [resolve(5,a,6,a)].
```

Figure 4: Prover9 displays output for the automated proof.

Because the number of clauses an automated theorem prover can generate greatly increases with respect to the size of the knowledge base, researchers have begun to form heuristics to evaluate the relevance of axioms when completing a proof. Some methods include evaluating the semantic similarity between predicates to determine which axioms might be more relevant when attempting to form a proof.

2.3 Semantic Similarity

Evaluating the similarity of two entities (i.e. classes or relationships) can serve as one heuristic when attempting to reduce the number of clauses generated during a proof. Multiple metrics have been developed for evaluating the semantic similarity of terms with different approaches, including: edge-counting measures, feature-based measures, and measures based on Information Content [**sanchez2012ontology**] [**rodriguez1999assessing**] [**roederer2009divvy**]. Edge-counting metrics for semantic similarity when applied to semantic hierarchies remain the focus of this work.

Calculating the distance between two entities in an ontology is a a straightforward and intuitive method of calculating the semantic similarity. One can formally define the metric as follows. In an undirected graph G defined as a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq (u, v)|u, v \in V$, one can define a path $path(a, b) = l_{1,...,}l_k$ as a set of links connecting a and b in a taxonomy and $|path(a, b)| = k$ as the length of the path [**sanchez2012ontology**]. One can calculate the semantic distance between a and b using equation **??** [**rada1989development**].

$$sim_1(a, b) = min_{\forall i}|path_i(a, b)| \tag{2}$$

Semantic hierarchies can be expressed as trees, and by incorporating depth of the taxonomy into the function, Zhibiao Wu has seen improvement in the metric. Because ontologies can vary greatly in depth due to the design of the ontology, some researchers have attempted to calculate semantic similarity using the lowest common ancestor (LCA), defined between two verticies a and b as the lowest vertex in the tree with both a and b as descendants (where we allow a vertex to be a descendant of itself) [**wu1994verbs**]. The *root* of a tree has no ancestors.

$$sim_2(a, b) = \frac{2 \times sim_1(LCA, root)}{sim_1(a, LCA) + sim_1(b, LCA) + 2 \times sim_1(LCA, root)} \tag{3}$$

3 APPROACH

This work aims to evaluate the effectiveness of using a semantic hierarchy generated from an ontology to calculate weights for predicates that will help focus the theorem prover on using axioms that are deemed more relevant to proving a conjecture. In efforts to quantitatively evaluate the effectiveness of the proposed methods, a series of experiments are conducted on multiple ontologies from the COmmon Logic Ontology REpository (COLORE)[1], a "testbed for ontology evaluation and integration techniques" [**gruninger2012specifying**]. Pellet, a semantic reasoner, is used to generate semantic hierarchies, which are then used to calculate the assigned weights for each predicate when executing proofs. Finally, tests were run using Prover9 to compare the default weights to the calculated weights. The effectiveness of the process was measured by comparing the number of clauses generated by Prover9 for each proof.

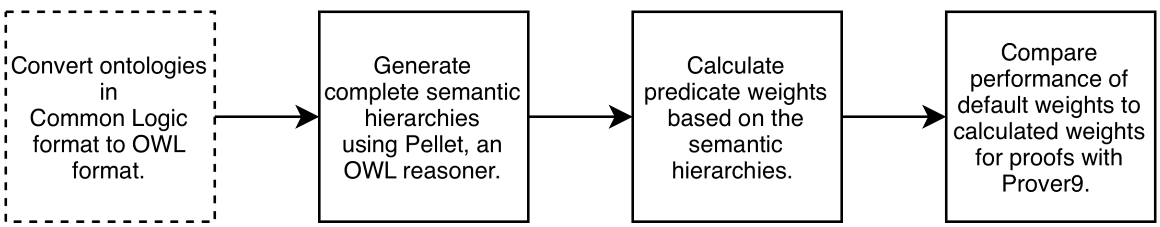

Figure 5: The figure above illustrates my process of conducting experiments. The first step, converting the ontologies into OWL format, lives outside of the scope of my research.

3.1 Converting Ontologies

The ontologies in COLORE are specified using the Common Logic syntax. No tools exist for generation of the complete hierarchy directly from an ontology defined using Common Logic. However, virtually all Web Ontology Language (OWL) reasoners efficiently implement the task of organizing classes. Thus, one needs to translate the ontology to OWL, use an OWL reasoner to complete the hierarchy, and then calculate predicate weight based on that hierarchy. Robert Powell has written a utility which executes the conversion and has generated the files necessary to conduct this research.

[1] `https://code.google.com/p/colore`

Not all ontologies in COLORE have conjectures, which limits the scope of my experiments to the sufficiently large ontologies.

3.2 Generating a Complete Heirarchy

After converting an ontology into the OWL format, one can generate semantic heirarchies including both asserted relationships and inferred relationships. Pellet can be used to generate the inferred semantic hierarchy, which are then displayed in Protégé (Figure **??**). The reasoner parses the axioms for inferred logical consequences (i.e. relationships between classes) not explicitly defined. The reasoner generates complete class hierarchies, but does not change the relationship hierarchy.

3.3 Calculating Weights for Resolution Theorem Proving

3.3.1 Default Weights in Prover9

Prover9 assigns weights to predicates automatically unless the user explicitly defines them. An understanding of the process helps one to develop new weights for the predicates. Lower weights give higher preference for a predicate when generating clauses. Rules for weighting axioms in terms of relevance when attempting to prove a specific conjecture are as follows [**mccune2005prover9**]:

- The default weight of a constant or variable is 1.
- The default weight of a term or atomic formula is one more than the sum of the weights of its arguments.
- The default weight of a literal is the weight of its atomic formula.
- The default weight of a clause is the sum of the weights of its literals.

Below is an example of how one may modify weights in Prover9 [**mccune2005prover9**].

```
list(weights).
  weight(a) = 3.                               % the weight of the constant a is 3
  weight(f(a,x)) = 5 * weight(x).              % weight(f(a,term)) = 5 * weight(term)
  weight(f(a,_)) = -1.                         % _ matches any variable
  weight(x | y) = 2 + (weight(x) + weight(y)). % add 2 for each "or" symbol
end_of_list.
```

3.3.2 Semantic Weighting Functions

Assigning weights to specific predicates will likely yield the best results. After semantic hierarchies have been generated, weights can be assigned to each class and subproperty. Two explicit weighting functions inspired by related works in calculating semantic similarity, were formed and tested. The weights for each conjecture were then calculated by hand and entered into a spreadsheet. A python script was used to generate the input files used by Prover9 from the spreadsheet for each conjecture. The calculated weights were then entered into Prover9 as additional input along with the axioms and the conjecture. The weighting functions are currently applied by hand to the ontologies, with the beginnings of an automated program underway.

3.3.3 Function 1

The first function attempts to make use of the completed class hierarchy generated by a semantic reasoner by giving preference to predicates existing on a path between pairs of predicates in the conjecture. For example, if one wished to prove the conjecture $Automobile(myCar)$ using the ontology provided in Figure **??**, it is reasonable to assume the theorem prover would need to traverse a series of axioms ascending the class hierarchy. Also, the 'Sedan' and 'Subaru Legacy' classes might not be given as much preference as predicates contained in the conjecture (i.e. $Automobile(x)$). Additionally, if the conjecture contains relationships (such as $Produces(x, y)$), once can apply the same principles. Additionally, in efforts to give lower preference to predicates not contained on paths connecting pairs of classes or relationships, unweighted ancestors and descendants are assigned higher weights. In order to achieve the goals above, Function 1 is defined as follows:

- Each predicate describing a class or relationship contained in the conjecture is given weight 1.
- For each pair of predicates describing classes within the conjecture, if a path exists between the two classes in the class hierarchy, predicates describing classes contained on the path are given weight 1.
- For each pair of predicates describing relationships within the conjecture, if a path exists between the two relationships in the relationship hierarchy, predicates describing relationships contained on the path are given weight 1.

- Decedents of predicates describing classes or relationships contained in conjecture without weights are given a weight corresponding to the depth of the entity. Subclasses and subproperties are given a weight of the respective parent class or property plus 1.
- All ancestors of predicates with a weight generated and all top-level classes corresponding to predicates without a weight assigned are given weight 10.
- All ancestors of predicates with a weight generated all top-level relationships corresponding to predicates without a weight assigned are given weight 10.

Given the example 'automobile' ontology and the conjecture $Automobile(myCar)$, the weights can be calculated as follows.

```
list(weights).
  weight(SubaruLegacy(x)) = 1.
  weight(Sedan(x)) = 1.
  weight(Automobile(x)) = 1.
  weight(Minivan(x)) = 2.
  weight(ToyotaSienna(x)) = 3.
  weight(FordWindstar(x)) = 3.
  weight(CarManufactuer(x)) = 10.
  % weight(Produces(x,y)) - This is not defined as the conjecture contains no relationships.
end_of_list.
```

3.3.4 Function 2

The second function attempts to make use of the lowest common ancestor (LCA) of each class or relationship with inspiration from equation **??**. Again, predicates in within the conjecture are preferred. Siblings and the parent of the LCA are weighted highly, and decedents of predicates without weights are given weights increasing with the depth of the semantic hierarchy.

- Each predicate describing a class or relationship contained in the conjecture is given weight 1.
- For each pair of predicates describing classes within the conjecture, if a path exists between the two classes in the class hierarchy, predicates describing the lowest common ancestor are given weight 1 and classes contained on the path which are neither a predicate contained in the conjecture or the lowest common ancestor are given weight 2.
- For each pair of predicates describing relationships within the conjecture, if a path exists between the two relationships in the class hierarchy, predicates describing the lowest common ancestor are given weight 1 and classes contained on the path which are neither a predicate contained in the conjecture or the lowest common ancestor are given weight 2.
- Siblings and parents of a LCA are given a weight 3.

- Decedents of predicates describing classes or relationships contained in conjecture without weights are given a weight corresponding to the depth of the entity. Subclasses and subproperties are given a weight of the respective parent class or property plus 1.

Given the example 'automobile' ontology and the conjecture $FordWindstar(myCar) \vee SubaruLegacy(myCar)$, the weights can be calculated as follows.

```
list(weights).
  weight(SubaruLegacy(x)) = 1.
  weight(FordWindstar(x)) = 1.
  weight(Automobile(x)) = 1. % LCA
  weight(Sedan(x)) = 2.
  weight(Minivan(x)) = 2.
  weight(ToyotaSienna(x)) = 3.
  weight(CarManufactuer(x)) = 3.
end_of_list.
```

4 EXPERIMENTS

4.1 Setup

Experiments were conducted using Prover9, an "automated theorem prover for first-order and equational logic" written by William McCune [**mccune2005prover9**]. Many tests were conducted using a version of the program which supports a Graphical User Interface (GUI), but a command line version, useful for running automated tests, exists. Git was used for version control and a repository containing source code for tests can be found at `https://github.com/stanleysmall/thesis`.

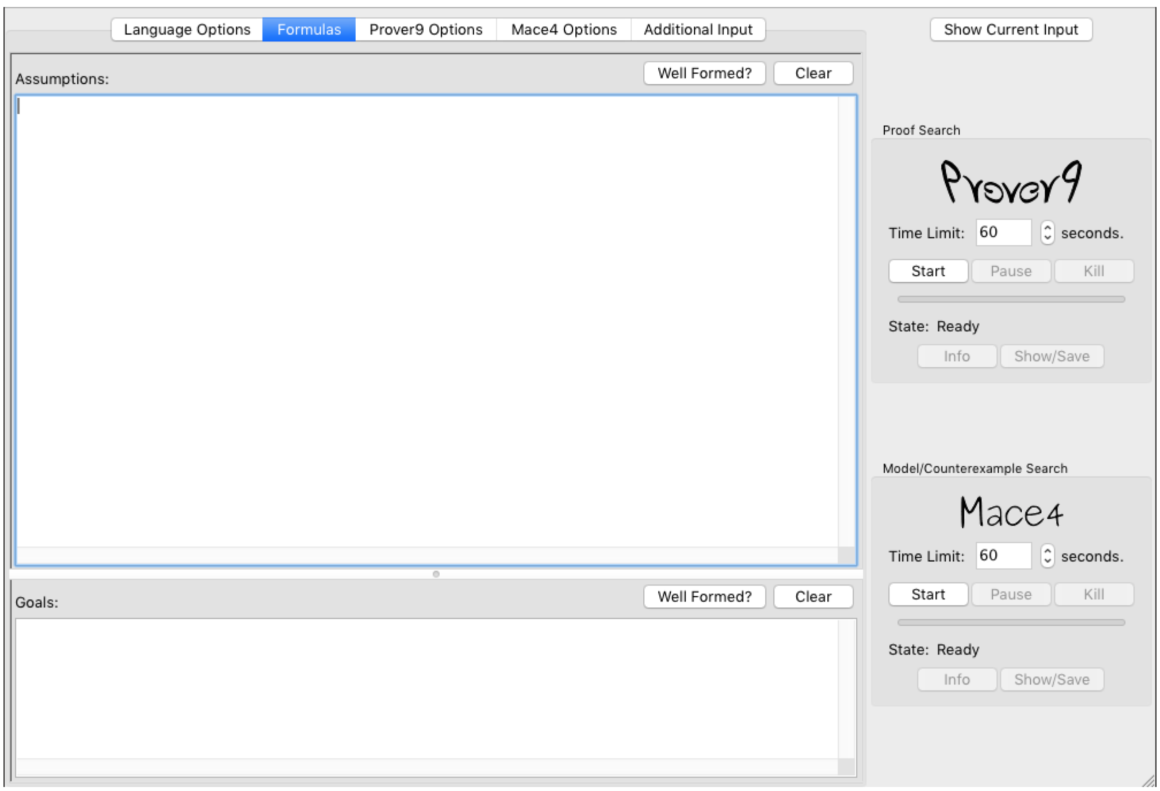

Figure 6: The GUI for Prover9 on macOS is shown above. One enters axioms into the top text field and conjectures into the bottom text field. Predicate weights are entered in the 'Additional Input' tab in the navigation bar. Users can start the proof by pressing the 'Start' button below the Prover9 logo after specifying a time limit. After the proof has completed, users can view the number of clauses generated by the proof by pressing the 'Info' button below the 'Start' button [**mccune2005prover9**].

4.2 Results

Below are empirical results for the described study. Each table contains a series of conjectures, and the number of clauses generated for both default weights and each of the weighting functions. A dash represents no change, and 0 indicates a change less than 1 percent. All weights for the tests are contained in Appendix A.

Conjecture	Default	Function 1	Percent Change 1	Function 2	Percent Change 2
1	85691	9165	-89	9231	-89
2	1803	1803	-	1803	-
3	1803	1803	-	1803	-
4	175	175	-	175	-
5	175	175	-	175	-
6	172	172	-	172	-
7	6357	6337	0	6225	-2
8	6015	5855	-3	2352	-61
9	1802	1802	-	1802	-
Average	11555	3032	-73	2638	-77
Median	1803	1803	0	1803	0
Sum	103993	27287	-74	23738	-77

Table 1: Results for the multidim_space_voids Ontology

Conjecture	Default	Function 1	Percent Change 1	Function 2	Percent Change 2
1	140734	50476	-64	50476	-64
2	480	754	57	742	55
3	295	295	-	234	-21
4	308	308	-	332	8
5	28188	28188	-	28188	-
6	11793	7830	-34	7830	-34
Average	30300	14642	-52	14634	-52
Median	6137	4292	-30	4286	-30
Sum	181798	87851	-52	87802	-52

Table 2: Results for the inch Ontology

Conjecture	Default	Function 1	Percent Change 1	Function 2	Percent Change 2
1	426	426	-	410	-4
2	285	285	-	285	-
3	426	426	-	430	-1
4	289	266	-8	324	-12
5	438	438	-	323	-26
6	283	240	-15	283	-
7	495	425	-14	255	-48
Average	377	358	-5	330	-12
Median	426	425	0	323	-24
Sum	2642	2506	-5	2310	-13

Table 3: Results for the multidim_space_physcont Ontology

Metric	Default	Function 1	Percent Change 1	Function 2	Percent Change 2
Average	13111	5347	-59	5175	-61
Median	459	432	-6	420	-8
Sum	288433	117644	-59	113850	-61

Table 4: Overall Results

4.3 Discussion

In some cases, the algorithm increases the number of clauses generated when proving a conjecture, but does not do so to the point where the proof does not finish. For the majority of proofs, the number of clauses generated decreases or remains unchanged.

Function 1 saw an average 19 percent reduction in the number of clauses generated and function 2 saw an average 23 percent reduction, suggesting the use of semantic hierarchies can reduce the number of clauses generated by an automatic theorem prover when attempting to prove a conjecture on an ontology. Both approaches seem to provide the same or better performance in many cases, with only a handful of cases where the number of clauses generated actually increased. Neither function performs significantly better than the other for a specific ontology, or for the majority of tests. The functions designed performed well for the ontologies tested; however, more tests are required to make generalized claims regarding the effectiveness of said methods.

The functions appear to help most when pairs of predicates are at a greater distance from one another (in regards to the minimum path), or when the semantic hierarchies are many levels deep. This seems to apply both to predicates describing classes and relationships, especially so when a percentage decrease in number of clauses generated for a specific conjecture exceeded 50 percent.

In the inch calculus ontology, conjecture 2 is `(all x all y (GED(x,y) & GED(y,x) & (all z (CH(z,x) -> CH(z,y))) -> CS(x,y)))`. In this case, the weighting functions increased the number of clauses generated by a significant amount. The relationship hierarchy is much smaller than those for the other ontologies. Additionally, two predicates are repeated twice in the conjecture, unlike many others. The combination of a shallow hierarchy and repeated predicates likely contributed to an increase in the number of clauses generated.

5 CONCLUSION

When proving specific conjectures with few predicates on large ontologies, semantic hierarchies can focus the search of a resolution theorem prover. Results of the experiments conducted indicate further work might yield lucrative results, especially for exceptionally large ontologies. Tests demonstrated success in a relatively unexplored domain of research.

5.1 Future Work

The scarcity of suitable ontologies to test provides many opportunities for advancement. Opportunities for further research include fully automating the search procedure, working with a larger number of ontologies to ensure the weighting functions actually do as they say, or developing a new approach for automatically weighting the predicates.

Adjustments regarding the tolerance or aggressiveness of the functions appears to be a promising path forward. Depending on the shape and depth of semantic hierarchies for an ontology, weights could be increased to values nearing 100 or more. Experiments conducted used a maximum weight of ten for any one predicate in efforts to reduce any increases in the number of clauses generated.

Table 5: multidim space voids weights for function 1

Entity	Type	Superclass(es)	1	2	3	4	5	6	7	8	9
CAVITY	Class	owl:Thing	10	10	10						10
Closed	Class	owl:Thing	10	10	10						10
ComplexV	Class	V	2	2	2						
Con	Class	S									
DPF	Class	F									
F	Class	”PEDΨ RPForDPF”									
Gap	Class	owl:Thing	2	10	10						10
HOL	Class	owl:Thing	2	10	10						10
Hole	Class	owl:Thing	10	1	10						10
ICon	Class	Con									
M	Class	”PEDΨ mat”									1
Max	Class	S									
MaxDim	Class	S									
Min	Class	S									
MinDim	Class	S									
NAPO	Class	POB									
PED	Class	POBorMorF	10	10	10						1
POB	Class	PED mat									
POBorMorF	Class	owl:Thing									
POBorMorRPF	Class	owl:Thing									
POBorRPF	Class	owl:Thing									
RPF	Class	F mat									
RPForDPF	Class	owl:Thing									
S	Class	owl:Thing	10	10	10						1
SimpleV	Class	V	2	2	2						
SimpleVorComplexV	Class	owl:Thing									
TUN	Class	owl:Thing	10	10	10						10
V	Class	SimpleVorComplexV	1	1	1						10
ZEX	Class	S									1
mat	Class	POBorMorRPF									1
BCont	ObjectProperty										
C	ObjectProperty					10	10	10	10	10	
Cont	ObjectProperty										
Covers	ObjectProperty										
DK1	ObjectProperty					10	10	10	10	10	
EqDim	ObjectProperty										
ICont	ObjectProperty										
Inc	ObjectProperty					10	10	10	10	10	
P	ObjectProperty										
PO	ObjectProperty										
PP	ObjectProperty										
SC	ObjectProperty										
TCont	ObjectProperty										
VS	ObjectProperty					10	10	10	10	10	
ch	ObjectProperty					10	10	10	10	10	
gt	ObjectProperty										
hosts	ObjectProperty										

Entity	Type	Superclass(es)	1	2	3	4	5	6	7	8	9
hostscavity	ObjectProperty										
hostscavityi	ObjectProperty										
hostscavityt	ObjectProperty										
hostsg	ObjectProperty										
hostsh	ObjectProperty										
hostshollow	ObjectProperty										
hoststunnel	ObjectProperty										
hostsv	ObjectProperty										
hostsve	ObjectProperty										
hostsvi	ObjectProperty										
lt	ObjectProperty										
r	ObjectProperty					10	10	10	10	10	
"="	ObjectProperty					10	10	10	10	10	
"gt="	ObjectProperty					10	10	10	10	10	
"lt="	ObjectProperty					10	10	10	10	10	

Table 6: multidim space voids weights for function 2

Entity	Type	Superclass(es)	1	2	3	4	5	6	7	8	9
CAVITY	Class	owl:Thing	10	10	10						10
Closed	Class	owl:Thing	10	10	10						10
ComplexV	Class	V									
Con	Class	S									
DPF	Class	F									
F	Class	"PEDΨ RPForDPF"									
Gap	Class	owl:Thing	1		1						10
HOL	Class	owl:Thing									10
Hole	Class	owl:Thing	1	1	10						10
ICon	Class	Con									
M	Class	"PEDΨ mat"									1
Max	Class	S									
MaxDim	Class	S									
Min	Class	S									
MinDim	Class	S									
NAPO	Class	POB									
PED	Class	POBorMorF	10	10							1
POB	Class	PED mat									
POBorMorF	Class	owl:Thing									
POBorMorRPF	Class	owl:Thing									
POBorRPF	Class	owl:Thing									
RPF	Class	F mat									
RPForDPF	Class	owl:Thing									
S	Class	owl:Thing	10	10	10						1
SimpleV	Class	V									
SimpleVorComplexV	Class	owl:Thing									
TUN	Class	owl:Thing	10	10	1						10
V	Class	SimpleVorComplexV	1	1	10						10
ZEX	Class	S									1
mat	Class	POBorMorRPF									1
BCont	ObjectProperty										
C	ObjectProperty					10	10	10	10	10	
Cont	ObjectProperty										

Entity	Type	Superclass(es)	1	2	3	4	5	6	7	8	9
Covers	ObjectProperty										
DK1	ObjectProperty					10	10	10	10	10	
EqDim	ObjectProperty										
ICont	ObjectProperty										
Inc	ObjectProperty					10	10	10	10	10	
P	ObjectProperty										
PO	ObjectProperty										
PP	ObjectProperty										
SC	ObjectProperty										
TCont	ObjectProperty										
VS	ObjectProperty					10	10	10	10	10	
ch	ObjectProperty					10	10	10	10	10	
gt	ObjectProperty										
hosts	ObjectProperty										
hostscavity	ObjectProperty										
hostscavityi	ObjectProperty										
hostscavityt	ObjectProperty										
hostsg	ObjectProperty										
hostsh	ObjectProperty										
hostshollow	ObjectProperty										
hoststunnel	ObjectProperty										
hostsv	ObjectProperty										
hostsve	ObjectProperty										
hostsvi	ObjectProperty										
lt	ObjectProperty										
r	ObjectProperty					10	10	10	10	10	
"="	ObjectProperty					10	10	10	10	10	
"gt="	ObjectProperty					10	10	10	10	10	
"lt="	ObjectProperty					10	10	10	10	10	

Table 7: inch weights for function 1

Entity	Type	Superclass(es)	1	2	3	4	5	6
ZEXI	Class	owl:Thing		10	10			
CH	ObjectProperty		1	1	1	1	1	1
CS	ObjectProperty		1	1	10	10	1	1
GED	ObjectProperty		10	1	1	1	1	10
INCH	ObjectProperty		1	1	1	1	1	1

Table 8: inch weights for function 2

Entity	Type	Superclass(es)	1	2	3	4	5	6
ZEXI	Class	owl:Thing		10	10			
CH	ObjectProperty		1	1	1	1	1	1
CS	ObjectProperty		1	1	10	10	1	1
GED	ObjectProperty		10	1	1	1	1	10
INCH	ObjectProperty		1	1	1	1	1	1

Table 9: multidim space physcont weights for function 1

Entity	Type	Superclass(es)	1	2	3	4	5	6	7
CAVITY	Class	owl:Thing							
Closed	Class	owl:Thing							
ComplexV	Class	V							
Con	Class	S							
DPF	Class	F							
F	Class	PED							
Gap	Class	owl:Thing							
HOL	Class	owl:Thing							
Hole	Class	owl:Thing							
ICon	Class	Con							
M	Class	”matΨPED”							
Max	Class	S							
MaxDim	Class	S							
Min	Class	S							
MinDim	Class	S							
NAPO	Class	POB							
PED	Class	owl:Thing							
POB	Class	”matΨPED”							
RPF	Class	”matΨF”							
S	Class	owl:Thing							
SimpleV	Class	V							
TUN	Class	owl:Thing							
V	Class	owl:Thing							
ZEX	Class	S							
mat	Class	owl:Thing							
BCont	ObjectProperty								
C	ObjectProperty		10						
Cont	ObjectProperty								
Covers	ObjectProperty								
DK1	ObjectProperty		10						
EQUALS	ObjectProperty								
EqDim	ObjectProperty								
ICont	ObjectProperty								
Inc	ObjectProperty		10						
P	ObjectProperty								
PO	ObjectProperty								
PP	ObjectProperty								
SC	ObjectProperty								
StrongC	ObjectProperty								
TCont	ObjectProperty								
VS	ObjectProperty								
ch	ObjectProperty		10		10	10	10	10	10
conporespace	ObjectProperty		10		10	10	10	10	10
convoidspace	ObjectProperty		10		10	10	10	10	10
dep	ObjectProperty		10		10	10	10	10	10
depcont	ObjectProperty								
depimmatcontains	ObjectProperty								
depmatcont	ObjectProperty								
detcont	ObjectProperty				1				
enclosesmat	ObjectProperty								
enclosesvoid	ObjectProperty						1		

Entity	Type	Superclass(es)	1	2	3	4	5	6	7
fullphyscont	ObjectProperty		10		10	10	10	10	10
gt	ObjectProperty								
hosts	ObjectProperty								
hostscavity	ObjectProperty								
hostscavityi	ObjectProperty								
hostscavityt	ObjectProperty								
hostsg	ObjectProperty								
hostsh	ObjectProperty								
hostshollow	ObjectProperty								
hoststunnel	ObjectProperty								
hostsv	ObjectProperty		10		10	10	10	10	10
hostsv1	ObjectProperty								
hostsv2	ObjectProperty								
hostsv3	ObjectProperty								
hostsvany	ObjectProperty		10		10	10	10	10	10
hostsve	ObjectProperty								
hostsvi	ObjectProperty								
immatcont	ObjectProperty				1				
inside	ObjectProperty							1	1
isurroundsmat	ObjectProperty								
isurroundsvoid	ObjectProperty								
lt	ObjectProperty								
matcont	ObjectProperty		1						
matdep	ObjectProperty		1		1				
matfillsinside	ObjectProperty								
matinside	ObjectProperty								1
matsplitinside	ObjectProperty								
osurroundsmat	ObjectProperty								
osurroundsvoid	ObjectProperty					1			
porespace	ObjectProperty		10		10	10	10	10	10
r	ObjectProperty		10		10	10	10	10	10
submaterial	ObjectProperty								
subvoid	ObjectProperty								
surrounds	ObjectProperty								
surroundsmat	ObjectProperty								
surroundsvoid	ObjectProperty					1	1		
voidinside	ObjectProperty							1	
voidspace	ObjectProperty		10		10	10	10	10	10
voidspaceall	ObjectProperty		10		10	10	10	10	10
"gt="	ObjectProperty								
"lt="	ObjectProperty								

Table 10: multidim space physcont weights for function 2

Entity	Type	Superclass(es)	1	2	3	4	5	6	7
CAVITY	Class	owl:Thing							
Closed	Class	owl:Thing							
ComplexV	Class	V							
Con	Class	S							
DPF	Class	F							
F	Class	PED							
Gap	Class	owl:Thing							

Entity	Type	Superclass(es)	1	2	3	4	5	6	7
HOL	Class	owl:Thing							
Hole	Class	owl:Thing							
ICon	Class	Con							
M	Class	”matΨPED”							
Max	Class	S							
MaxDim	Class	S							
Min	Class	S							
MinDim	Class	S							
NAPO	Class	POB							
PED	Class	owl:Thing							
POB	Class	”matΨPED”							
RPF	Class	”matΨF”							
S	Class	owl:Thing							
SimpleV	Class	V							
TUN	Class	owl:Thing							
V	Class	owl:Thing							
ZEX	Class	S							
mat	Class	owl:Thing							
BCont	ObjectProperty								
C	ObjectProperty		10						
Cont	ObjectProperty								
Covers	ObjectProperty								
DK1	ObjectProperty		10						
EQUALS	ObjectProperty								
EqDim	ObjectProperty								
ICont	ObjectProperty								
Inc	ObjectProperty		10						
P	ObjectProperty								
PO	ObjectProperty								
PP	ObjectProperty								
SC	ObjectProperty								
StrongC	ObjectProperty								
TCont	ObjectProperty								
VS	ObjectProperty								
ch	ObjectProperty		10		10	10	10	10	10
conporespace	ObjectProperty		10		10	10	10	10	10
convoidspace	ObjectProperty		10		10	10	10	10	10
dep	ObjectProperty		10		10	10	10	10	10
depcont	ObjectProperty								
depimmatcontains	ObjectProperty								
depmatcont	ObjectProperty								
detcont	ObjectProperty				1				
enclosesmat	ObjectProperty								
enclosesvoid	ObjectProperty						1		
fullphyscont	ObjectProperty		10		10	10	10	10	10
gt	ObjectProperty								
hosts	ObjectProperty								
hostscavity	ObjectProperty								
hostscavityi	ObjectProperty								
hostscavityt	ObjectProperty								
hostsg	ObjectProperty								
hostsh	ObjectProperty								
hostshollow	ObjectProperty								

Entity	Type	Superclass(es)	1	2	3	4	5	6	7
hoststunnel	ObjectProperty								
hostsv	ObjectProperty		10		10	10	10	10	10
hostsv1	ObjectProperty								
hostsv2	ObjectProperty								
hostsv3	ObjectProperty								
hostsvany	ObjectProperty		10		10	10	10	10	10
hostsve	ObjectProperty								
hostsvi	ObjectProperty								
immatcont	ObjectProperty				1				
inside	ObjectProperty							1	1
isurroundsmat	ObjectProperty								
isurroundsvoid	ObjectProperty								
lt	ObjectProperty								
matcont	ObjectProperty		1						
matdep	ObjectProperty		1		1				
matfillsinside	ObjectProperty								
matinside	ObjectProperty								1
matsplitinside	ObjectProperty								
osurroundsmat	ObjectProperty								
osurroundsvoid	ObjectProperty					1			
porespace	ObjectProperty		10		10	10	10	10	10
r	ObjectProperty		10		10	10	10	10	10
submaterial	ObjectProperty								
subvoid	ObjectProperty								
surrounds	ObjectProperty								
surroundsmat	ObjectProperty								
surroundsvoid	ObjectProperty					1	1		
voidinside	ObjectProperty							1	
voidspace	ObjectProperty		10		10	10	10	10	10
voidspaceall	ObjectProperty		10		10	10	10	10	10
”gt=”	ObjectProperty								
”lt=”	ObjectProperty								

AUTHOR'S BIOGRAPHY

Stanley C. Small grew up in Hampden, Maine with his mother Diane and his father Scott. He attended the University of Maine and received a Bachelor of Science degree in Computer Science in May of 2019.