Verifying arbitrary safety-related rules using Web Ontology Language

VISHNU SURESH RAJU
Verifying arbitrary safety-related rules using
Web Ontology Language

VISHNU SURESH RAJU
Abstract

This project work has been undertaken in order to explore the possibility of verifying arbitrary safety-related rules in the context of heavy vehicles subject to ISO26262 functional safety standard of vehicle correctness using semantic web reasoning techniques that are in Linked Data format. The aim is to further use this as a method to claim functional safety for different configuration of vehicles, in a highly automated way. The ability of current system of tools to perform the verification involves manual work and is difficult to perform because of the size and complexity of the data.

The entire work was studied and implemented within Scania, where in the integrated data from system safety department, in the Linked Data format was used for the implementation of the tool. The project work was proceeded in two stages. The initial stage of the project was surveying the existing reasoners and their applications to different problems in verification of rules, on the basis of different comparison criteria’s and benchmark results. The second stage of project involved determining a suitable way to represent the rules, in order to verify them against the available data.
Sammanfattning

Detta examensarbete har genomförts för att undersöka möjligheten att verifiera godtyckliga säkerhetsrelaterade regler i samband med tunga fordon som omsättes av ISO 26262 funktionell standard för fordonssäkerhet, med hjälp av semantiska webresoneringsmetoder i länkat dataformat. Syftet är att använda detta vidare som en högt automatiserad metod för funktionell säkerhet för olika fordonskonfigurationer. Det nuvarande systemet med verktyg för att utföra verifieringen innebär manuellt arbete och är svårt att använda på grund av datas storlek och komplexitet.

Examensarbetet utfördes inom Scania, där data tillhandahölls av system-säkerhetsavdelningen. För implementering av verktyget användes länkade data. Arbetets första steg var att kartlägga de befintliga resonerarna och deras tillämpningar på olika problem vid kontrollen av regler baserade på olika jämförelsekriterier och benchmarkresultat. Den andra etappen av projektet var att bestämma ett lämpligt sätt att representera reglerna för att verifiera dem mot tillgängliga data.
## Contents

1 Introduction .................................................. 1
   1.1 Semantic Web and Linked Data ......................... 2
       1.1.1 Semantic Web Stack .......................... 3
   1.2 Problem Statement ..................................... 4
   1.3 Research Methodology .................................. 5

2 Background .................................................. 7
   2.1 Description Logic ...................................... 7
       2.1.1 TBox ........................................... 8
       2.1.2 ABox .......................................... 9
       2.1.3 Reasoning ..................................... 9
   2.2 Expressing Rules in OWL ............................... 10
   2.3 Protégé: The Tool ...................................... 11

3 Related Work ................................................ 14
   3.1 Initial Approach ...................................... 14
   3.2 Reasoners .............................................. 15
       3.2.1 Non-DL Reasoners ............................ 16
       3.2.2 DL Reasoners ................................. 16

4 Representation Of Rules ................................... 19
   4.1 OWL- Web Ontology Language and Semantic Reasoners ... 19
       4.1.1 Manchester Syntax ............................ 20
   4.2 Use Case Study based on specification and structuring of safety requirements ................................................. 20

5 Design and Implementation ................................. 24
   5.1 Model for the system .................................. 24
       5.1.1 Setup ......................................... 24
   5.2 Preliminary Analysis and proceedings ................... 25
Chapter 1

Introduction

In most areas of enterprise, there is an increasing need for useful information, especially in big companies, with the increase in volume of data generated. The increasing complexity results in information that might contain much more heterogeneous and distributed data for each domain. In order to analyse such information in a holistic way, one might need access to multiple, if not all types of data. This brings up the need for an effective way for Data Integration\(^1\).

Linked Data\(^2\), a novel approach based on Semantic Web Technology, is a promising way to integrate all data related to a particular domain together. This approach provides a way to publish structured information, by connecting different data resources by means of relationship. One of the characteristics feature of this technology is defining an ontology, which helps in providing meaningful definition for the objects that are related to a particular domain, in a formal way. Such a formal definition of the different resources, help Semantic Reasoners [15] (which is a piece of software to infer logical consequences from set of asserted facts), to analyse the information better.

A goal of this thesis, is to explore the possibility for creating a tool to verify arbitrary safety related rules, subject to ISO26262 standards for vehicle correctness using the reasoning technique. This way of verification helps in claiming functional safety on different configuration of vehicles in a highly automated way which otherwise requires a lot of manual work. Claiming of functional safety corresponds to ability to provide evidence on a variety of things, for example, for each hazard, a corresponding safety goal is defined in each product configuration. Such a kind of verification would require a

\(^1\)https://www.ibm.com/analytics/data-integration
\(^2\)https://www.w3.org/standards/semanticweb/data
lot of understanding and review of documentation with the current system. Automating this, would reduce a lot of manual work and provides a much better way for analysis of the information.

1.1 Semantic Web and Linked Data

Semantic Web Technology \(^3\), defines a set of standards to support "Web Of linked data", that contains a large number of resources that are interconnected. It follows a standard model of data interchange of the web called Resource Description Framework (RDF)[16] which defines statements in the form of \(<s><p><o>\):

\(<s>\) - subject
\(<p>\) - predicate
\(<o>\) - object

\(<s><p><o>\) expressions, which is also referred as the RDF-triple. The predicate shows the relationship which connects the other two entities, subject with the object. An example for this kind of relationship in natural language is given below:

**Truck has wheels**

Here the predicate *has* relates the subject *Truck* with object *wheels* forming a relationship between the two distinct data objects. Linked Data follows four main principles for publishing data out of it, as coined by Tim Berners Lee which are mentioned below:

1. Use URI (Unified Resource Identifier) as name for the resource.
2. Use HTTP URI so that users could look up these resource.
3. Each URI should provide some useful information about the resource.
4. Create more URI so that users could discover additional information about the resource.

[^3]: https://www.w3.org/standards/semanticweb/
Given the basic principles of linked data, an example for an RDF-triple could be given in the form of:

\[
<\text{uri://scania_rm#Truck}> <\text{http://scania/has}> <\text{uri://scania_rm#Wheels}>
\]

The above triple is equivalent to the statement Truck has Wheels, except that its definitions have been changed to that of URI. Each entity within this RDF-triple could be given as:

subject: <uri://scania_rm#Truck>
predicate: <http://scania/has>
object: <uri://scania_rm#Wheels>

Here as we can see the subject which is under consideration (<uri://scania_rm#Truck>), is related by the predicate (<http://scania/has>) to the object (<uri://scania_rm#Wheels>). The reasoning over the linked data helps to obtain more complete answers or inferences based on these formal definitions given to data resources. In order to get a better understanding on reasoning could be performed over the Linked Data, its better to understand the Semantic Web Stack, which is mentioned in following section.

1.1.1 Semantic Web Stack

The Semantic Web stack Figure 1.1\(^4\), illustrates the underlying architecture of the structure followed by the Semantic Web. In this section, a brief introduction is given to different layers within the architecture, the details of which are given below.

The URI or the Unified Resource identifier at the very bottom level uniquely identifies each resource. On top of this, the RDF or the Resource Description Framework is defined which is standard model for data interchange over the Web as explained in the previous section.

The stack also defines a querying language called SPARQL, which helps in retrieving information from the a triple store containing RDF-triple.

On top of the RDF, the vocabulary languages RDF-Schema(RDFS), and Web Ontology language (OWL) which provides additional constructs for defining richer semantic to the data resources. While RDFS provide a basic vocabulary language for the RDF data, OWL provides a much richer vocabulary. An example for defining RDFS relationship could be <uri://scania_rm#Truck> <rdfs:subclass> <uri://scania_rm#Vehicles>

\(^4\)https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html
where the property `<rdfs:subclass>` helps in enriching `Truck` with more information by relating it with `<uri://scania_rm#Vehicles>`. Similarly another simple example for an OWL relationship could be defined as `<uri://scania_rm#Truck> <owl:equivalentClass> <uri://scania_rm#Vehicles>` which is another way of defining relationship between the different data entities.

Reasoning, which comes in the Unifying Logic part of the architecture and is done on top of the vocabulary languages helps in making the implicit knowledge more explicit. Assuming that there exist two triple of the form:

`<uri://scania_rm#Truck> <rdfs:subclass> <uri://scania_rm#Vehicles> <uri://scania_rm#Vehicles> <rdfs:subclass> <uri://scania_rm#HeavyVehicles>`, then the reasoning over these set of triples, helps in inferring the third axiom that `<uri://scania_rm#Truck> <rdfs:subclass> <uri://scania_rm#HeavyVehicles>`.

### 1.2 Problem Statement

The thesis aims to answer the following research questions:
• What are the different OWL based reasoners and how to compare them in terms of the expressiveness and overall correctness in reasoning over varying ontologies?

• Given rules expressed in OWL, how can the data related to the rule be transformed from RDF to OWL and then verified against the rule?

In the context of vehicle safety, verification of the vehicle correctness is of much importance for ensuring the overall safety of any vehicle. This involves verifying that only meaningful products are created and also ensuring that design for the products are done in a proper way. Besides internal procedures, there are standards like the ISO26262 standards, which help in defining the best practises for such kind of verification.

Consider a simple scenario of such a kind of verification where in there is a Requirement that "If the brake pedal is pressed, the vehicle should decelerate" which applies to all Trucks that are produced. Verification of rules is a way to ensure that triples of the form: 

This example was given to give an intuition on the kind of problem that we are trying to solve with this thesis work, but the real world data and the rules which needs to be verified are much more complex. Also it is a problem to verify these kinds of rules because all data that are related to testing and data related to different kinds of Requirement that apply for different configurations of vehicles, were all present in different tools. Once all these information are integrated in Linked Data format, verifying the rules in this way helps in automating the process and reducing the manual work needed in this process.

The evaluation for the thesis work involves a creating a tool that could accept arbitrary rules in natural language, which needs to be verified against a specific database. The database used for thesis, is Scania dataset from the system safety department which contains integrated data from different tools in the Linked Data format(details in Chapter 4).

1.3 Research Methodology

With many semantic web reasoners available based on different ontology languages, identifying a suitable reasoner in the context of linked data is an im-
important task. This is essential in the context of ensuring the correctness on the results inferred by the reasoner.

The initial approach to solving the problem involved obtaining a deeper understanding on the semantic web and the need for reasoning on these information. The documentation [5] and online documentations\(^5\)\(^6\) gave an initial understanding on the underlying architecture of Semantic Web which gave an insight into exploring the different knowledge representation languages available like Description Logic, First Order Logic etc. As OWL2 based reasoners are very much related to Description Logic, this was determined to be the language to be used for the future works of this project work, which is covered in Chapter 2.

Further literature study for the thesis was to identify a suitable OWL 2 based reasoner that could be used within the context of linked data. This involved much more deeper understanding as well as analysis of the different reasoners. Different surveys, benchmarking results for OWL reasoners, journals and papers comparing different reasoners needs to be studied in depth to make a better analysis of these reasoners which are presented in Chapter 2.2.

Finally, there was a need to formulate a way for expressing the rules and the required information in right way in order to make the right inferences out of it. The tool Protégé\(^7\) and a guide\(^8\) turned out to be very useful for familiarizing and understanding ontologies better. An iterative approach for forming the rules and testing was adopted which is presented in Chapter 4. Chapter 5 covers details on Design and Implementation of the tool. Results are presented in Chapter 6 followed by a Chapter 7 on Discussion and Conclusion.

\(^5\)https://www.w3.org/standards/semanticweb/
\(^6\)https://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html
\(^7\)https://protege.stanford.edu/
\(^8\)http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLtutorialP4_v1_3.pdf
Chapter 2

Background

This section is split into two sections. The first section describing Description Logic and some key elements used within the language. The second section describing. The second section introduces the background study that was needed to identify ways to represent the rules, in order to perform the reasoning. The section also briefly describes the a tool Protégé, which helped in understanding the initial study on ontology.

2.1 Description Logic

Knowledge representation had been one of the primary areas of focus in the area of artificial intelligence in order to represent information about the world in a way that could be understood by the computer. One of the core feature of any such form of representation is its expressiveness, as in how effectively a statement could be expressed in a logical way, maintaining the correctness of the argument. Description Logic is one such family of formal knowledge representation languages. This section covers some of the common concepts used within them.

As [1] points out, the initial approach of network based representation of knowledge, using nodes and links had been the primary motivation for further researches in the area of Description Logic. Description Logic (DL) is a family of languages for knowledge representation for the knowledge within the particular application domain taken that is taken into consideration. DL enables definition of the knowledge base to reason about the content in the particular domain. It models concepts, roles and individuals and their relationships. A concept within a particular domain represents the set of individual objects and links that used to characterize the relationships between them. Concepts also
have some simple properties related to them. An *terminological axiom* is used to define a logical relationships between the roles or concepts or in short, make statements on how concepts or roles are related to each other. These axioms are of the form $C \equiv D$ or $C \sqsubseteq D$, where $C$ and $D$ are concepts. DL also supports different kinds of *role restrictions* like the *existential quantification* and *value restriction* which helps in defining ways to form concepts like "individuals having a yellow car" or "individuals all of whose cars are yellow" which could be represented as $(\exists \text{hasCar} \cdot \text{Yellow})$ and $(\forall \text{hasCar} \cdot \text{Yellow})$ respectively. In order to distinguish the concepts better, the individual objects that refer to the second argument of such restrictions are referred as *role fillers* to those roles, for example, in this previous example, Yellow would be the filler to the role hasCar. Another example for the value restriction could be given as $\forall R.C$, which refers to set of all individuals that are in the relationship $R$ with the concept being defined belonging to the concept $C$. Thus we could interpret the concepts as a set of individuals and roles as sets of pairs of individuals.

A distinction that could be seen evidently in a database will be the intensional knowledge and the extensional knowledge. By intensional knowledge, one means the general knowledge about a problem and the latter which is specific to a problem. Correspondingly the Description Logic Database is divided into the $TBox$ and $ABox$. The $TBox$ contains the intensional knowledge and the $ABox$ contains the extensional knowledge. A more detailed information about each of these are given below.

### 2.1.1 TBox

$TBox$ is one of the key elements of the DL knowledge base and is composed of operations that are used to build a terminology within the database. One of the basic form of declaration is the concept definition where previously defined concepts are used to define new concept. A basic example of this is given below:

$\text{Man} \equiv \text{Person Male}$. $TBox$ are also referred as the intensional knowledge in the form of terminologies, which are built through declarations that give more details to them. This is one of the basic form of representation for the definitions, but are stronger than the ones used in other kinds of representation of knowledge. As specified in the work, there are some common assumptions which are usually made about the DL terminologies which are listed below:

- only one definitions is allowed for a concept name
• definitions should be acyclic, which refers to the fact that concepts should not be defined in terms of itself or other concepts that indirectly refer to them

Classification, which essentially is the process of placing a concept in accordance with the hierarchy specified is one of the primary tasks for constructing the terminologies.

2.1.2 ABox

The *ABox* basically tries to include the extensional knowledge for a particular domain of interest, i.e., the assertions about the different individuals also called as a membership assertions. A very basic for the definition of this kind of assertion is given below:

• Female ⊓ Person(ANNA)

The introduction of the different individuals and asserting the properties for them, specifies one important aspect of it. These ways of assertions could be classified as: *concept assertions* and *role assertions*. By *concept assertions* one try to state that a particular individual belong to a concept and with the *role assertions*, one try to assign the filler for a particular role with an individual as such.

2.1.3 Reasoning

Some commonly used types of inferences that could be done over a dataset are:

• **Subsumption:** One basic inference on concept expressions is in determining if a particular concept always denote a subset of the set denoted by another one, which is broadly described as subsumption. Determining the *subsumption* is the problem of checking if a particular concept denoted as $D$ is a more general concept of another concept defined as $C$, where the concept $D$ is referred as the *subsumer* and the concept $C$ is defined as *subsumee*.

• **Satisfiability:** is another common inference on concept expressions where in the reasoning is done to check if a concept expressions necessarily
denote an empty expression or not. In simpler terms, this could be explained as way to check if a knowledge base is meaningful or not. This could be broadly described as a special case of subsumption where the subsumee is an empty set.

- **Realisation** Realisation is a kind of reasoning where the reasoner tries to identify the implied instance or type of relationships between named individual and named classes within a particular domain. Only direct relations between the two are considered for these kinds of reasoning.

**Reasoning Algorithms**

Broadly the DL based algorithms could be classified into two - *Subsumption algorithms and Tablaeu based algorithms*. Subsumption based algorithms also referred as the structural subsumption Algorithm, compares the syntactic structures of concept description to infer new results. Tableau based algorithms turns out to be useful in those cases that require further more expressivity in defining the conditions. The details of each of these algorithms are given below. As described above, the subsumption based algorithms which take into consideration the syntactic structures, are found to be more efficient but with lesser expressivity as these DL systems do not allow for negation and disjunction operation providing. These algorithms proceed in two different phases where during the initial stage the description that is to be tested for subsumption are normalized and then the syntactic structure of the normal forms obtained are compared. Tableau based algorithms fills up this limitations of the subsumption algorithms. Instead of testing directly on the concept description, these algorithms use the negation to reduce subsumption to satisfiability of the concept description.

**2.2 Expressing Rules in OWL**

In this section, the possibility for defining a suitable ontology, which as we recall from Chapter 1, is a formal naming and structuring of knowledge is explored. Such a formal definition of the information is needed, in order for the reasoner to make inferences.

In an initial study to understand the current systems that follow, [27] turned out to be very useful. The paper primarily focuses on defining OWL DL ontologies to determine the relationships in the area of domain engineering. The work gave an understanding on the feature modelling as well as how the ver-
ification could be done using OWL ontologies. In the initial stage, the tool Protégé\ig{https://protege.stanford.edu/} was used in in defining the ontology and running the reasoner to make inferences. This helped in understanding and also finding a way to express rules in later stages for the project work. This work also focuses on ways of writing rules that could be run on top of these ontologies and also points out on how the different features of reasoning could be use up efficiently by limiting the definitions of ontologies within TBox. Also other research works like [23] gave some insights into ways on how to create the rules that are relevant to infer the required information from the data that we have.

A detailed documentation [18] from the same team of Protégé, gave an explanation on the usage of different constructs that are used in the context of Description Logic in Manchester Syntax\ig{https://www.w3.org/TR/owl2-manchester-syntax/}. The syntax as such was much more simpler and readable way to represent the axioms. The same syntax\ig{https://www.w3.org/TR/owl2-manchester-syntax/} used while setting up the interface for the final tool that was created for this project work.

### 2.3 Protégé: The Tool

Protégé, is a free open source platform to construct domain models and knowledge based applications with ontologies. It was developed at Stanford Centre for Biomedical research. The version of Protégé 5.2 was used in the initial stages of the development of ontologies for this project to understand and get better understanding on defining ontologies with respect to the rules that we were trying to check. A brief introduction on the tool will be given below.

**Class Hierarchy** [18] is the most important part of the ontology creation where we define the different classes as subClasses or superClass of one another, which brings up relations between those classes. The tool provides a Graphical User Interface to organize the different classes as required. We could also define the hierarchy of the all the classes if the structure for such a hierarchy is known. **Object** and **Data** properties[18] are the next most important part of the ontology creation where we define and associate properties between the classes. **Object property** Properties tries to connect the different individuals or objects within the domain. For example, the object property parentOf can be used to represent the parenthood relationship between individuals. On the other hand, **data property** tries to connect individuals with a
literal value. We could also define rules for the classes in the *subClass* of section within the class hierarchy which helps in classifying the different classes which follows the rules accordingly. **Annotation Property** [18] could be used to add comments and additional information regarding the different ontologies.

The above mentioned features within Protégé forms the very basic ways to create ontology by defining the classes and properties among them. The most important feature, which is the **reasoner**, is used to interpret or deduct meaningful information from the already created axioms. Different plugins are available, which could be used with the Protégé, like Fact++, Hermit, Pellet that could be used effectively. An attempt was made to see if the reasoner Konclude could be used within Protégé indirectly by using a Konclude server and using OWLHTTP plugin to use the reasoner. But as per the guidance of the development team of Konclude, a more stable reasoner like Fact++ was decided to be used for further evaluation and reasoning for this project.

When it comes to writing rules within the tool, it supports different sets of predefined terms that could be used for creating axioms for defining the rules that certain set of classes follow within the defined Ontology. Some of those predefined terms are listed below  

1. **only**- This keyword could be used to define axioms or rules of the type *AllValuesFrom*, that relates the property with another class or an individual. A formal definition could be given as:

   \[ (\forall R.C)^I = \{ a \in \Delta^I | \forall b. (a, b) \in R^I \rightarrow b \in C^I \} \]

   For example, the class expression *hasChild only X*, is a way to denote the set of all classes of individuals, if they are related to any other individual by the *hasChild* property, then the other individual must be of type X.

2. **some**- This keyword within the tool is used to define axioms of the type *SomeValuesFrom*, that relates a property with another individual by means of *atleast* relationship. A formal definition for it could be given as:

   \[ (\exists R.T)^I = \{ a \in \Delta^I | \exists b. (a, b) \in R^I \} \]

---

4For all the formal definitions we consider Interpreations *I* (where a concept is interpreted as set of individuals and roles are interpreted as sets of pairs of individuals), \( \Delta^I \) is the non empty set of domain of interpretations, \( R^I \subseteq \Delta^I \times \Delta^I \), \( R \) is the atomic role for binary relations \( R^I \subseteq \Delta^I \times \Delta^I \), \( C, D \) are concepts
An example of this could be given as hasChild Some X, which is the class of individuals which are related to at least some other individual of type X.

3. **exactly** This keyword is a way to denote the classes of individuals that are related to exactly one other individual by means of the property. One additional care that one must take into consideration while using this property is the open world assumption, that is the assumption that the truth value of a statement may be true irrespective of whether or not it is known to be true.

4. **and** and **or** The operators **and** and **or** are also the other sets of commonly used predicate to connect different sets of classes to one another. The **and** denotes the intersection and **or** denotes the union of the two class expressions. A formal representation of them could be given as:

- **and**- \((C \cap D)^I = \{C^I \cap D^I\}\)
- **or**- \((C \cup D)^I = \{C^I \cup D^I\}\)

Besides the above mentioned restrictions on the sets of data, there needs to be some additional care taken in defining the meaning of the class expression to be formed, either the subclass of expressions or the equivalent relationships based on the needs for the different types.
Chapter 3

Related Work

The section explains how the literature study was carried out, beginning with some related works on the topic, followed by the study on some reasoners based on different languages.

3.1 Initial Approach

In this project work, the primary focus is on identifying a way for defining an application to perform the semantic reasoning, in the context of safety requirements for the fuel level display in vehicles [28]. Different technical papers, journals and open documents were referred to understand the related works to the topic. In the search for understanding the current works related to the topic, the work [8] turned out to be good starting point for it. In this work, the authors tried to propose a methodology using OWL (Web Ontology Language) to define safety related requirements and defining a prototype for the proposed methodology to test case of Lane departure warning system. The authors also gave an insight into different issues faced during the work as well pointed on the issues that might occur due to the informal representation of the natural language text. The work uses OWL ontologies for the formal specification of the domain knowledge which is built upon open world assumption providing better flexibility.

The work [6] was an experimental study to give a deeper understanding on issues that might occur during the mining or extraction of information out of the linked data, where the authors showed the challenges involved in reasoning over data by conducting different experiments, and showed on how it failed to derive an explanation from the defined knowledge.
3.2 Reasoners

In an effort to study the different reasoners, a study on both DL based as well non-DL based reasoners were performed. Before determining the right reasoner which could be used in the course of this work, there was a need to understand more about the currently available reasoners in the context of linked data. The work in [13][14] was the starting point for this, which contained a comparative study on the different available reasoners. While the work [13] was survey over 35 actively used reasoners directly addressing the reasoner developers over different reasoner characteristics involving reasoning services, expressivity level and completeness of the implemented algorithm, the work [14] survey semantic web reasoners and their relationship to different knowledge representation languages. These studies were a good starting point towards understanding more on how a comparative study could be made on the reasoners. Apart from these works, the results from 1 also gave an insight into the different disciplines used to make study on the OWL reasoners.

Also [20] addressed further on the review and comparative study on the rule languages and inference engine suited for a particular application. This work explored the different engines based on the expressiveness, supported built in functions, inference algorithms and presented the results for the different reasoners and rule languages taken into consideration. Different other benchmarking results like were also taken verified of which [19] was the recent ones which made a comparative study on the rule based inference engines. This study involved three different reasoners, the Jena inference engine, the Euler YAP Engine(EYE) reasoner, and the BaseVisor reasoner. The results showed different trends for each of the criteria that were taken into consideration which was varying for different criterias like the load time reasoning time. The resources in RDF data, which form the basis of Semantic web, could be also represented in different serializations like the RDF/XML or Notation3. The work [4] gave more insight into issue of using Notation3, which mainly focuses on using Implicit Quantification. Although the paper proposes a solution for this, it involves the overhead of conversion of the RDF data which could incur further loss in the performance of the reasoning.

3.2.1 Non-DL Reasoners

In an effort to explore and understand more on the available reasoners were analysed. (Hai H. Wang, Jin Song Dong)\(^2\) mainly used a formal tool Alloy which had its own reasoner, Alloy Analyzer to reason over the rdf data. The reasoning approach defined in the work explains on creating a semantic model in Alloy and using it to reason over the semantic web. The paper also focuses on the point that Alloy is a first order declarative language based on relations and relations form the basics in the formation of the Semantic Web or the Linked Data. But one primary limitation as pointed out in this work is the scalability of the reasoner, Alloy Analyzer, although its one novel approach used for the purpose. Another work [22] tried to point out some of the limitations of Description Logic, to introduce an approach for using Alloy Analyzer for reasoning over Semantic Web. The major limitations of DL reasoners as pointed out in this work are the inefficiency of DL to reason complex properties ontologies, inability to perform instance level reasoning. The paper also defines the approach used to create the Alloy model which could be used later on with the Alloy Analyzer and reason over it.

Prolog based reasoners, like the EYE [26] was yet another reasoner which was studied. As the work shows, the performance offered in this case, is far more superior when compared to OWL DL based reasoners. One of the core part of the EYE reasoner is an Abstract Machine which accepts N3 (Notation 3) notation, which is a Prolog representation of parsing an RDF data. Although these notations are another representations of the basic data over which the data is working, there are some issues that needs to be addressed as the one mentioned in this paper which mainly focuses on the individual interpretations of the and renders it difficult to make clear statements on the relation to other formal languages like the Description Logic or the first Order Logic. Another issues which might be a minor with the context might be its incompatibility as of now with tools like Protégé, which makes it rather cumbersome in defining ontologies and setting the reasoner for the Semantic Web.

3.2.2 DL Reasoners

The studies on reasoning over semantic web and the work [1] suggest that high quality ontologies are crucial for the semantic web and their construction, integration and evolution greatly depends on the availability of a well defined

\(^2\)https://pdfs.semanticscholar.org/077e/80bab91c20d24597faec8980047b5cbb887a.pdf
semantics and powerful reasoning tools. Description Logic [2], which is one such formalism for knowledge representation of the ontologies turned out to be one such ideal candidate for the this providing both. Some previous works [7] had claims questioning the trade off or the mismatch between the expressivity and efficiency in the reasoning power. But as the work [1] suggest, the gap for this mismatch has reduced over time. The work has also focused on the general semantics for defining the ontologies using the Description Logic as well.

As mentioned above [19][20] as well taking into consideration some other studies like [11] on the OWL based reasoner, its clear that the performance results are quiet varying in each study based on the ontology supported by each reasoner. As the paper suggested, there are few reasoners which provided complete reasoning support. In further attempt to identify semantic reasoner in the context of correctness, the Konclude reasoner [24], a novel reasoner developed in 2014 was studied in detail which gave some good results in some recent study as well.

**Konclude reasoner** [24], is a high performance reasoner based on tableau logic calculus of Description Logic. The reasoner is designed to provide performance and provide optimization technique even using multiple CPU’s at several levels of its architecture. The work [24] also support its claim from the results on comparative study on some commonly used OWL 2 reasoning system. Although the Konclude reasoner did not outperform the other reasoners in the classification time, the reasoners Fact++ [25], Pellet [21] and HermiT [9] which had errors in classifying 280, 318 and 260 ontologies which was handled by Konclude stating that classification performance of Konclude is better than the other 3 reasoners. A similar comparison on the consistency check was also performed which showed a better performance for Konclude over the others. The paper also points about the ability of the reasoners to use multiple threads to improve the performance as another added advantage of the reasoner. The work [17] is another interesting results from ORE which shows the results from different tracks when compared to other competing reasoners. The results shows that Konclude reasoner performs well in almost all the categories. Konclude being a novel reasoner, in its initial stages of development, was able to show some valid results and hence will be chosen as suitable reasoning technique.

The next task having defined the rule language would be to write the rules for define the ontologies reason based on that. The work [12] defined a starting point that helped in defining the rules in description logic. Taking the com-
parative studies from above into consideration, Protégé \(^3\) was found to be an efficient tool for defining ontologies as well as use reasoners upon. Another important work[18] that helped in understanding and defining rules is the one which gave as starting point. This work took an example of pizza ontology in defining and explaining the different reasoning mechanism that could be used along with the tool. The examples helped in much more deeper understanding of the very basic concepts and the different usage. The work [3] gave further evidence in defining and representing the rules which was related to biomedical engineering.

As Protégé does not provide a way to integrate it with a triple store, there was a need to create a tool that could be used to read in information in suitable format, define an ontology out of it and run the reasoner based on the information provided within the tool. The OWL-API[10] helped in defining an ontology, from the triple store containing huge number of triples and run a reasoner on top of the ontology thus created.

\(^3\)https://protege.stanford.edu/
Chapter 4

Representation Of Rules

In this section, a very brief introduction is given on OWL or Web Ontology language, followed by the use case study which is taken into consideration for verification of the rules. The design and implementation will be explained in detail in Chapter 5.

4.1 OWL- Web Ontology Language and Semantic Reasoners

Web Ontology Language (OWL)\(^1\) is a logic based Semantic Web language, used to formally represent rich and complex knowledge about data resources, as well as define the relations between them. The current version of OWL, OWL2 \(^2\) will be used in this work to represent the rules. OWL 2 based ontologies helps in defining a formal naming and structuring of the information, that are related to a particular domain. Such a kind of formal definitions, would help with the semantic reasoners to infer logical consequences out of it. OWL2 is defined by a five core documentation which describes the structural specification, mapping to RDF graphs, Direct Semantics, RDF based semantics and conformance requirements. Different reasoners based on OWL2 DL were studied the details of which will be explained in Chapter 2. OWL 2 also supports different syntaxes like OWL 2 functional syntax, OWL2XML syntax, Manchester Syntax of which the Manchester Syntax\(^3\) will be used in this thesis work which is more readable and easy to understand when compared to

\(^1\)https://www.w3.org/OWL/
\(^2\)https://www.w3.org/TR/owl2-overview/
\(^3\)https://www.w3.org/TR/owl2-manchester-syntax/
other syntaxes.

4.1.1 Manchester Syntax

Manchester Syntax is a user friendly syntax for defining OWL 2 ontologies. In this work, Manchester Syntax will be used as a primary way to represent rules, so that individual resources and properties could be parsed correctly and then later on read from the triple store. Also, the rules represented in this syntax could be parsed directly using ManchesterOWLSyntaxParser\(^4\) which can parse once all the entities mentioned in the rule are available in the ontology. The Table 4.1 shows the different OWL constructs and its equivalent Manchester Syntax and DL syntax.

\[
\begin{array}{|c|c|c|}
\hline
\text{OWL Constructor} & \text{DL Syntax} & \text{Manchester OWL} \\
\hline
\text{intersectionOf} & C \sqcap D & C \text{ and } D \\
\hline
\text{unionOf} & C \sqcup D & C \text{ or } D \\
\hline
\text{complementOf} & \neg C & \text{not } C \\
\hline
\text{someValuesFrom} & \exists R.C & R \text{ some } C \\
\hline
\text{minCardinality} & \geq n.R & \text{min } n R \\
\hline
\text{maxCardinality} & \leq n.R & \text{max } n R \\
\hline
\text{value} & \exists R.o & R \text{ value } o \\
\hline
\end{array}
\]

4.2 Use Case Study based on specification and structuring of safety requirements

This section introduces the use case study which was taken into consideration for verification of rules for this work assuming that all necessary information are available.

The Figure 4.1 gives the structuring of the safety requirements with respect to ISO26262\(^5\) specifications. Here each entities of the diagram has a special meaning with respect to specification, SG denote SafetyGoal, FSR denote

\(^4\)http://owlcs.github.io/owlapi/apidocs_4/org/semanticweb/owlapi/util/mansyntax/ManchesterOWLSyntaxParser.html

\(^5\)https://www.iso.org/standard/43464.html
CHAPTER 4. REPRESENTATION OF RULES


![Diagram of requirement structuring](image)

**Figure 4.1:** Structuring of Requirement adapted from [28]

Also the different lines connecting each requirement has some meaning, while the lines with an arrow denote the *fulfill* relation, the other one denotes the *assumedBy* relation connecting each entity. All the different rules that we are trying to check is with respect to these relations that are connected to one another. Based on the requirements for the safety ISO26262, the tasks were broken down into smaller ones for easiness in representation as well as interpretations. As per the ISO26262, the entire requirement sets could be subdivided into 4 different safety requirements—Functional, Software, Hardware and Technical to which all the sets of vehicles should adhere to. Apart from the above mentioned four, another requirement namely safety Goal forms the top in the hierarchy which specifies if a particular vehicle is safe or not. Apart from this, each of these requirements should also adhere to the Automotive Safety Integrity Level (ASIL) requirement based on the set of concepts defined and will be explained in the end of this chapter.

The statements for the different rules as well as the rules defined for each case are listed below:

1. *Each SafetyGoal is fulfilledBy some FunctionalSafetyRequirement.*

   By this rule, the aim is to determine all the *SafetyGoal* that are *fullyFulfilledBy some FunctionalSafetyRequirement*. This is equivalent to Figure 4.2 representation, where the closed curve in blue color denotes the set of all *SafetyGoals* and curve in red color shows all the *Requirements* that are *fulfilledBy some FSR* and region common to both curves is the set of all *SafetyGoal fullyFulfilledBy some FunctionalSafetyRequirement*.
In protege the same could be represented as: \textit{fullFilledBy some FunctionalSafetyRequirement} which is equivalent to $\exists$ \textit{fullFilledBy. FunctionalSafetyrequirement}.

2. Another challenging tasks was to represent the requirements such that each of the requirement had to fulfil the ASIL requirements. When it comes to describing the ASIL, the rules need to be written correctly such that if some requirement fails to follow the exact requirement, it should be reported as inconsistent based on the fullFilledBy relations that it has to follow.

3. All SafetyGoals do not fulfill another SafetyGoal.

4. All SafetyGoals do not fulfill or are assumed by other Requirements

5. All Non SafetyGoal Requirements either fulfill or are assumeBy other Requirements.

A suitable way to represent the rules 1 to 5 needs to be determined. Since Description Logic looks for explicit axioms, in order to infer certain results, a suitable way need to be determined the concept of No Requirements. In order to determine the if a particular resource has a \textit{fulfills} relation or not, there is a need to traverse through all the related statements before confirming if it has that relation or not. Same method was used to relate each resource with the \textit{assumes, fulfilledBy or assumedBy} relations.
Besides these rules, all these requirements need to follow the **Automotive Safety Integrity Level (ASIL)** validation, which is a risk classification scheme as defined in the ISO26262 specification, classifies the different requirements to different classes like ASIL QM ASIL A, ASIL B, ASIL C and ASIL D. While ASIL D indicates the highest integrity requirement, ASIL A shows the lowest. Each of these ASIL are related to one another in the hierarchy by means of certain criteria, which means, if a particular requirement belongs to ASIL B, then that requirement need to be *fulfilledBy* a combination of two requirements of ASIL A or at least one ASIL B. The same applies to other ASIL’s, for example, if a requirement belongs to ASIL C, then that requirement needs to be *fulfilledBy* either another ASIL C, or a combination of at least an ASIL B and ASIL A or more than three ASIL A. In short, if we assign each ASIL value some integer values like A=1, B=2, C=3, D=4 and if a requirement belongs to a particular ASIL, then that requirement should be *fulfilledBy* at least a combination that adds up to it, example D=(B+B) or D=(A+A+A+A) and so on.
Chapter 5
Design and Implementation

This section could be split into different sections. The first section contains an introduction to the model for the system where the new tool comes into place. The next section explains further the preliminary analysis and proceedings taken for the project. The final two subsections explains the implementation specific API’s and schematic representation for the proposed solution for this work.

5.1 Model for the system

5.1.1 Setup

If we recall the rules mentioned in Chapter 4, we are trying to verify rules of those formats against the linked data within the system safety department at Scania. This dataset contains information obtained from different tools, integrated together in a single database. Each of these URI’s used help in uniquely identifying the resources from the database. A small snippet from the dataset is shown below:

```
```

- **Test Data Set:** Linked Data Set from System Safety Department, at Scania
- **Libraries Used:** storeAPI, OWL API, Hermit reasoner (OWL-API)
This dataset contains information in the RDF data format saved in the knowledge base which needs to be used for the semantic reasoning as well. The Figure 5.1 shows the basic structure on which the proposed idea is to be implemented. The RDF triple store contains the integrated linked data from the different sources. The proposed linked data reasoner is mentioned as Semantic Web Reasoner in the bottom part of the Figure and needs to be implemented in two different steps.

During the first step, the task is to verify a set of predefined rules (Recall from Chapter 4) over the linked data that we have. Once we are able to check the rules, then we need to determine a way to write the rules in a better way so that the different resources and properties could be identified and read from the triple store to run the reasoner. With respect to Scania dataset, we use a set of subroutines definitions in order to query data from the triple store.

### 5.2 Preliminary Analysis and proceedings

As per the instruction from the developers of Konclude and also due to the incompatibility of the reasoner with OWL-API which was determined to be used for defining ontologies, a more stable reasoner was recommended for proceeding for this project work. The reasoner HermiT was used for the future works which gave somewhat consistent performance for varying ontologies. Another advantage would be the easiness to embed this reasoner with the tool Protégé, which is one of the commonly used and efficient tool for
reasoning over ontologies.

The next task on deciding the reasoner was to understand and to write rules more clearly and precisely as per the requirement. The rules were to be defined as per the ISO26262 requirements which involved inferencing safe vehicles from the list of all vehicle configurations possible. Some of the related works as mentioned in the previous chapter (Chapter 3), helped in framing a rule for the final implementation of the tool.

The use case study for the system requirement for the fuel level display, as we recall from Chapter 4, was taken into consideration. This contained different sets of requirements, Functional, Hardware, Software, Technical and Safety Requirements connected to one another based on the ISO26262 standards.

The linked data from the Scania System Safety department was taken for the analysis. The database as such contained the different requirements that were linked to one another by means of their relationship based on the safety requirement of the fuel display system.

The proceedings for the creation of the ontologies on top which reasoner could be run, is defined as following. In the current implementation of the prototype for the application, the user is expected to write the rules in correct syntax of the Description Logic following the grammar. The tool ANTLR\textsuperscript{1,2} was used to define a grammar for the language which followed the same syntax as that of the Manchester syntax. The grammar for the valid set of accepted expressions are defined in the Table below.

The definition of the grammar further helped in parsing each entity correctly, i.e., to identify the properties and the resources based on the OWL syntax. Once the entities were identified a get request is made using a storeAPI to fetch the required information which are then added to the ontology defined earlier. Also in the current implementation, an iteration is made through all the fetched resources to identify if they are related to the resource based on which the details are added into the ontology. Once all the details are available within ontology, we could run the reasoner using the routines available using the OWL-API. The axiom needed for the reasoner to run, is created by using the \texttt{DLParser} which creates axioms once all the details are available within the ontology.

In order to perform the ASIL validations, the users will have mention the set of classes to be checked along with the rules that they try to check the axiom as well.

\textsuperscript{1}http://www.antlr.org/
\textsuperscript{2}https://tomassetti.me/antlr-mega-tutorial/
5.3 API for Accessing Linked Data

Scania has built an API that was used for accessing the linked data resources that are present in the triple store, called the storeAPI. The storeAPI exposes an API for storing and retrieving of the linked data resources in a RESTful (Representational State Transfer) way and also offers additional capabilities for updating and deletion of the resources from triple store. By RESTful, we actually mean that it uses the existing protocols, like HTTP GET, POST etc. to perform the different operations. storeAPI as such provided a greater flexibility and easiness in access of the specified resources as well as certain relationships that connects the resource to one another.

5.3.1 Operations

storeAPI as a whole is dependent on some other libraries of Lyo store \(^3\) is an important one. Lyo Store is a library that is to cache the resources for faster access of information from the triple store. In order to use the API to access the triple store data, there needs to be one local copy of the triple store databases being set up. Some of the commonly used operations using storeAPI for this application and its syntax are mentioned below:

GET

The GET request is used to GET a particular resource from the triple store. The storeAPI needs to be run as a server before trying to fetch the required data using it.

QUERY

The QUERY request is used to QUERY a particular resource from the triple store. The storeAPI needs to be run as a server before trying to fetch the required data using it.

\(^3\)https://github.com/eclipse/lyo.store
5.4 Using the OWL API for Java integration

OWL API \(^4\) is a java API and reference implementation that could be used for creating, manipulating and serialising OWL ontologies. There are different components within the API with the latest version supporting reasoner interfaces in order to work with reasoners like Fact++, Hermit, Pellet and Racers. The IDE eclipse was used to develop the program where the work helped in configuring the API correctly within the IDE.

One primary task in the project work was to set up all the dependencies as needed for the work correctly. This include the storeAPI, Lyo store, OWL API to perform the reasoning tasks. Once the environment was set up correctly, the next task was to configure read the data correctly from the triple store and create axioms correctly so as to run the reasoner on top of the ontology.

The *exactly* clause was used in defining the relationships between the different requirements. For example, the *fsr_fuel fulfilledBy exactly 1 fsr_coo* stated the *fsr_fuel was fulfilledBy exactly 1 fsr_coo*. It also helped in the deriving inferences for ASIL validations for the requirements as well as other restrictions that were to be tested for the particular use case study.

As mentioned before, the tool ANTLR was used to identify properties and resources as separate entities while reading the rule from a text file. Once those were identified, suitable axioms need to be created out of the information fetched. In order to run the reasoner, an ontology need to be defined and created first, which contains all the basic information needed to run the reasoner upon. For testing purpose, the rule *SafetyGoal fullFilledBy FunctionalSafety Requirement* was taken into consideration. As triple store does not contain the statement in the form of axioms, the axioms need to be correctly defined before which they are created and added into the ontology.

In order to ease the implementation using the OWL API, a helper class was defined at first, which provided most of the functionalities provided by the API, like defining OWL Class, OWL Properties, creating axioms for *some, only* and *exactly* clauses as well as basic reasoning service like the classification to retrieve the subclass of a particular super class defined. This reduced the task of reusing a certain set of commonly used sets of axioms that were needed for the development process.

Further, the creation of the different properties like *fulfilledBy, assumedBy, NotRequirement* etc were also separated into different classes. The axiom to create *NotRequirement* was created in need for checking a particular rule and

\(^4\)http://owlapi.sourceforge.net/
the axiom was created based on the number of properties that it was referenced to, for eg, if a particular class did not fulfill any other requirement, then an axiom was created stating that. The reason for creating axioms like this is due to *open world assumption*, ie, what is not known to be true is unknown. This means, there should be a way to explicitly state it so that the reasoner could infer new results out of it.

Now coming back to simple example rule stated above *(SafetyGoal fullFilledBy FunctionalSafety Requirement)*, the tool fetches all the SafetyGoals, like SafetyGoal1, SafetyGoal2..., and tries to check if there is *fulfilledBy* relation for them. If there is such a relation, then the details are added to the ontology and in case if there is no such relationship, then the axiom *fulfilledBy exactly 1 NotRequirement* is added to the ontology. A small portion of the ontology showing the same is shown below.

This is small portion from the ontology that shows axioms related to *safetyGoal_1* the the how the different axioms are formulated while executing the rule. Now when the rule *(SafetyGoal AND (fulfilledBy some FunctionalSafetyRequirement))* is executed to the ontology containing these axioms, the reasoner could infer *safetyGoal_1* as one of its results. This is because, *safetyGoal_1* is a *SafetyGoal* and is also *fulfilledBy fsr_coo* which is a *FunctionalSafetyRequirement*. 

<!— http://rdf.scania.com/dl_reasoning/safetyGoal_1 —>
5.4.1 Language for the Rules

Once the different helper classes were defined, there was a need to create and find a way where the users should be able to write the rules in an effective way which could be checked against the triple store. Although the OWL API provided some in build parsers like the ManchesterSyntaxParser, TurtleSyntaxParsers (which are ways to define the same set of data), it cannot be used for the current context as all these parsers needed the complete knowledge base. Thus we needed a parser to be defined separately that could read in the contents from a file and identify the different resources as well as the properties connecting them. For the easiness in reading and understanding, the ManchesterOWLSyntax was taken as a way to write the rules and a parser was created with the help of ANTLR \(^5\) for this purpose. It helped in simplifying the process as the software helped in tokenizing and creating a parser through which we could visit and add the resources as they are identified. The parser supports all the basic syntaxes of ManchesterOWLSyntax, the explanation for the each which are given below:

1. some - equivalent to the `someValuesFrom` relation in the OWL
2. only - equivalent to the `allValuesFrom` relation in the OWL
3. AND - equivalent to the `intersectionOf` relation in the OWL
4. OR - equivalent to the `unionOf` relation in the OWL
5. not - equivalent to the `ComplementOf` relation in the OWL
6. min - equivalent to MinimumCardinality relation in the OWL

For the easiness to understand, the different rules that needs to be checked using the tool could be represented as in given below.

1. \((\text{SafetyGoal } \text{AND} (\text{fulfilledBy some FunctionalSafetyRequirement}) \text{AND} (\text{fulfills some (not Requirement)}) \text{AND} (\text{assumedBy some NotRequirement}) \text{AND} \text{fulfilledBy some(testedBy some (testResult value true))}) \text{AND} (\text{ASILA OR ASILC OR ASILB OR ASILD}))\)

2. \((\text{SafetyGoal } \text{AND} ((\text{fulfilledBy some NotRequirement}) \text{OR} (\text{fulfills some (FunctionalSafetyRequirement OR TechnicalSafetyRequirement))}))\)

\(^5\)http://www.antlr.org/
Here the rule 1, is trying to list all those Safety Goals that fulfilledBy some FunctionalSafetyRequirement and not assumedBy or fulfill any other requirement and should also belong to a correctly classified ASIL (either A, B, C or D) which is an intersection of the rules which we are trying to check. The tool works under the assumption that the each of the resources mentioned in the above statements are present in the form of rdfs:class within the triple store and also there exists some resources that are subclasses of each of the resources mentioned in the statements.

A simple explanation for the rule 1 written above is that, it tries to infer the set of all safety goals that are fulfilledBy (that are having a fulfilledBy relation) some FunctionalSafetyRequirement. These SafetyGoals should also follow the other two parts mentioned in the rules (by using the intersection property ), ie, it does not fulfill or assumes any other requirements in effect. In this way the first rule helps in checking for the rules and giving results that follow these rules correctly.

The second rule (rule 2) is a way of checking for those SafetyGoals that fail with the rule 1 which is represented in the using the rule language as shown.

5.5 Proposed Solution

The Fig.5.2 shows the general flow diagram for the tool right from where the rules are written in text file to the creation of OWL-API Objects which could be used for later processing.

Once the rule is written in the text file, the parser identifies the different object properties and resources that are represented in the form of Description Logic. Once the tool identifies each entity, all the subclass of the entities are fetched in the form of Jena model\(^6\). The Jena Model is an ontology model that presents RDF as ontology classes, individuals, properties and so forth so that it could be used by the OWL API to make inferences on the retrieved data.

The information retrieved as Jena Model, are identified as Jena Ontology classes, which are mapped with reference to rdfs:class as specified in the triple store. The identification of the different Ontology classes helps in further navigating through the different statements of the fetched information as well as identifying the subjects with respect to it. We now create the OWLClass which are the OWL API equivalent for the OWL classes out of the subjects that are fetched and add those to the ontology. Creation of the OWL Class itself does

---

\(^6\)https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Model.html
not add to any information to the ontology and they need to be associated to different object properties or data properties as defined by the different statements of it. This is done by traversing each statement and adding the details if the required properties are found. Once the OWLObject are created and the required informations are added to the ontology we could make the reasoner run over the ontology by creating suitable class expressions.

In the tool, once the required informations are added to the ontology, we use the `DLParser`, which is an parser provided by the OWL-API to further create an DL axiom out of it and feed into the reasoner. The reasoner on reading all the necessary information needed for the reasoning, applies the reasoning in order to fetch the `subClass` of the result class which contains the DL axioms to be checked. All the `subClass` for the result will be the required classes or the outputs for the rule that we are trying to check.

In the Chapter 6, the results obtained on creation of such a tool for verification of rules are mentioned.
Chapter 6

Results

This section broadly describes the results from the thesis work and its further discussions. The initial approach introduces what preparations were done before proceeding to development of the tool. Following section describes the results from the tool and the reason for obtaining the results.

6.1 Initial Approach

The project work was explored in different stages, starting from the tool, Protégé which helped in understanding Description Logic and on how reasoners help in making inferences.

As an initial attempt to understand the requirement structure and to familiarize with the Description Logic, the tool Protégé was used. As Protégé provided some built in reasoners within the tool, it helped in getting a better understanding on how the different requirements could be structured and the rules should be written. The tool also gave further insights into ways in which the rules could be represented in knowledge representation languages, without losing the intended meaning of each statement. Overall it helped in understanding Description Logic more and forming axioms for later stage in the development of the tool.

6.2 Using Scania DataSets for use case study

As explained in previous section, the use of Protégé and the different reasoners in its context gave an initial attempt on how the reasoners could be run and rules could be written. The next task was to incorporate the same with more
on a practical scenario and to verify that it is possible to represent the rules in OWL in order to make inferences on Linked dataset.

If we recall from Chapter 5, two different API’s were used to read the necessary information and then form the ontology. The store API was used to access the Linked Data set within the Scania. The OWL API was used to define the classes as well as properties for the different sets of classes similar to the way it was done in Protégé. The OWL API also provided some in-built support for reasoners like Hermit which was used in the initial sets of reasoning and testing as there was no complete and full support for the reasoner Konclude. With all these configurations defined, the basic support provided by the system was to check the rules against the data set given.

### 6.2.1 Representing rules to be verified

As a starting point for the implementation of the tool, it was decided to perform only the verification of rules within the tool, without actually specifying the rule. This was later on changed where the user will be able to write in rules in a text file in the Manchester Syntax for the Description Logic which will be checked against the linked Data set. The tool took in inspiration from Protégé in defining the keyword for the different restrictions, like some, only, or and and. Apart from these predefined ones, an equiv keyword was also defined to relate the equivalentTo relationships.

The different steps involved involves sequential steps, as mentioned in introduction of Chapter 6.2. The rules which are to be verified are represented in Manchester Syntax, within a text file. The tool basically creates an ontology from the different entities mentioned in the rules. Finally the reasoner is used to infer the results from the ontology, based upon the rules written. The below two rules (recall from Chapter 5) were formed to verify that the implementation of the tool.

1. \((\text{SafetyGoal AND (fulfilledBy some FunctionalSafetyRequirement)} \text{AND (fulfills some (not Requirement)) AND (assumedBy some NotRequirement)} \text{AND fulfilledBy some(testedBy some (testResult value true))) AND ASILA OR ASILC OR ASILB OR ASILD)}\)

2. \((\text{SafetyGoal AND ((fulfilledBy some NotRequirement OR (fulfills some (FunctionalSafetyRequirement OR TechnicalSafetyRequirement)))}})\)

The Rule 1, infers all those SafetyGoal, that are fulfilledBy some FunctionalSafetyRequirement and that does not fulfill any other Requirement and all
that are fulfilledBy some Requirement that was testedBy a testCase, that had value true. Besides this, the SafetyGoal should also belong to ASIL classification, which is implied by the OR statement within the rule. On the other hand, Rule 2, infers all those SafetyGoals that fulfills some (FunctionalSafetyRequirement OR TechnicalSafetyRequirement) and is not fulfilledBy any Requirement.

6.2.2 Results from the Tool

As we recall from Proposed Solutions in Chapter 5, the rules mentioned in previous section are written in a text file for implementation of the tool. The tool, reads in content from the triple store, and creates an ontology out of it. Finally the reasoner is used to make inferences on the ontology created, using the rules mentioned in text file.

The Figure 6.1 shows the results after executing the rules as run in the eclipse¹ console. The results show the inference made by the reasoner, based on the rules written on the file. Here the rule:1 as mentioned in the console output, equates to the Rule 1, mentioned in Section 6.2.1. The same holds for the Rule 2 in the Figure 6.1.

In order to further understand, on the results shown in the Figure 6.1, we can look in to visual representation of the Linked data that was used for this work. For this purpose, a tool GraphDB² was used. GraphDB helps in providing a visual representation with all the relationships connecting a particular resource with others within a Linked data structure. The Figure 6.2 shows such an illustration for one the resources obtained as results from the tool implemented, which is safetyGoal_1. This is a helpful tool to visualize the various relations connecting the different entities in the triple-store and could be done by mentioning the URI of the resource. It also represents the relations that

¹https://www.eclipse.org/
²http://graphdb.ontotext.com/
bonds the resources together. The illustration further helps in asserting why the reasoner made such an inference on the ontology formed.

The Figure 6.2 shows the various relations that are connecting the safetyGoal_1 and other entities within the triple store, and the keyword mentioned on top of the arrow mark mentions the name of the relationship that binds them together. In this figure, we can see that safetyGoal_1 is fulfilledBy fsr_coo where fsr_coo is a FunctionalSafetyRequirement which is also a Requirement. Also safetyGoal_1, does not fulfill any other Requirement and it belongs to one of the ASIL classification, which is mentioned on the right of the Figure. Comparing this with Rule 1 mentioned in previous section, it satisfies all these conditions individually. This being the reason that the reasoner inferred safetyGoal_1 as one of its results during the execution of the rule.

Now, in order to understand, why any other SafetyGoal did not satisfy the conditions for Rule1, we will look into visual representation for the resource safetyGoal_2 shown in Figure 6.3. Here we could see that, even though safetyGoal_2 is fulfilledBy some FunctionalSafetyRequirement, it also fulfills another FunctionalSafetyRequirement. As a result it fails to satisfy condition for Rule 1 and is not inferred by the reasoner. But this turns out to be the reason, why safetyGoal_2 is one of the inferences made for Rule 2, ie, since safetyGoal_2 fulfills a FunctionalSafetyRequirement, one of the conditions become true and as result it turns out to be one of the inferences made by the reasoner for verifying Rule 2.
The Rules 1 and 2 presented in the result are very simple verifications performed by the tool. But the results show that, it is indeed possible to make inferences, by forming the right rules in OWL and with a triple store that contains enough information on each resource in Linked data format. The tool as such presents the results as expected, by using the rules mentioned within the text file to form an ontology and making inferences on it.
Chapter 7

Discussion

This section details on what could have been done differently and also on why some decisions were made and could be broadly divided into two subsections. The first subsection, Evaluation discusses the results of the work and how it answered the research question and the second subsection explains more on limitations and possibility for future works.

7.1 Evaluation

If we recall the results from Chapter 6, the current implementation of the tool could verify arbitrary safety related rules against triple store that was used for evaluation of the work. The rules which needs to be verified is to be written in Manchester Syntax in a text file. The tool reads the rule which was written in the file and fetches the relevant information from the triple store to create an ontology. Finally, the reasoner is used to verify the rules over the ontology formed. The results also explains the reason for inferences made by the tool, while verifying the rules against the ontology.

There were two main research questions which the thesis tried to answer, if we recall from Chapter 1. The first research question was - What are the different OWL based reasoners and how to compare them based on expressiveness and how to compare them based on expressiveness and correctness in reasoning? In order to answer this question, a comparative study was performed on different types of reasoners mentioned which is mentioned in detail in Chapter 3. Out of this study and also from benchmark results of ORE 2015, Konclude reasoner is found to be one of top performing OWL reasoner.

The second research question was - Given rules expressed in OWL, how can the data related to the rule be transformed from RDF to OWL and then
verified against the rule? In order to answer this question, a tool was created that could verify rules written in OWL against the triple store which contains data in Linked data format. Chapter 4 discusses on the particular use case study that was taken into consideration to verify the results and Chapter 5 discusses on model for the system along with the proposed solution. Also, the results from this work (in Chapter 6) shows how the rules were represented in OWL, and how that rule was used in creating an ontology that could be used by the reasoner to interpret the results. This in turn shows that, it is possible to represent the rules in OWL and it is possible to verify arbitrary rules against the Linked data.

7.1.1 Challenges

There were some significant challenges at different phase of the thesis. The foremost of this was forming the rules in the right way and logical manner so that the results obtained were consistent with the Linked data used for evaluation. As Description Logic in general is based on the concept of open-world assumption, everything that is not stated explicitly are not considered true. This made creation of some rules a little indirect in its representation, for example, in order to check the rules of the form SafetyGoals cannot fulfill another requirement, the tool now tries to traverse the entire list of statements to ensure that there is no fulfill relation connecting the resource to another and then add the axiom manually to the ontology so that the reasoner could identify and confirm if the rule exists or not and hence make the inference as well.

Another challenge was to integrate all the different components used for building the tool, and to have the required results obtained. This include setting up the Api to fetch the information from triple store and also trying to reason on why some reasoners could not be setup the way its expected to.

7.2 Future Work

There are limitations for this work as well. As of now, the tool uses Hermit reasoner for reasoning. This execution time and efficiency of reasoning could be improved with Konclude reasoner as from the studies shown in the related works. But as the reasoner is in the initial stages of development, there are some compatibility issues of this with the OWL API, on which the developers are working on right now. The reasoner as such is a parallel, high performance reasoner which can take advantage of multiple cores within a shared memory
environments. Once fixed, the reasoner as such could perform well with the current tool.

There are still possibilities to extend the tool with respect to the architecture being followed. If we recall from Chapter 5, Fig 5.2 represent how the OWL Objects are created from the triple store data. The model could be improved provided some changes are made in the triple store with respect to the data. This include, if there is a way to represent different entities like SafetyGoal as owl:class and properties like fulfills as Object properties within the triple store, then we could minimize the possibility of conversion from the Jena model to Ontology model and then the OWL Objects, as OWL API also provides way to directly create ontology out of the strings of information fetched from the triple store.

Another possibility for improvement of the tool is with respect to the development of a Domain Specific Language (DSL) for the tool based on the parser that was created using the tool. The current implementation works on the assumption that the rules are written correctly within the text files in the correct form of the ManchesterOWLSyntax. The language could give provision for auto suggestion of the certain keywords that are specified within the parser rules. This helps the users further in framing the rules and running the reasoner more smoothly.

The use of Konclude reasoner could bring in more parallelism as well as performance with respect to the reasoning capabilities as it uses the multiples cores within a shared memory environment. The execution time as such depends on the number of entities that needs to be fetched from the triple store and time needed for reasoning is very less when compared to the overall execution time. There are some additional possibilities for further optimizing this so as to reduce the total inferring time of the tool as such.

### 7.2.1 Sustainability ethics and societal impacts

The content presented in this thesis does not provide anything additional on top of Linked data nor reasoners that could be considered changes to their ethical or societal impacts. It could also be argued that, a system that makes verification of rules in a highly automated way, which otherwise would require lot of human intervention, reducing the risk for possible errors and efficient use of resources. The results presented in this thesis heavily supports this and makes the verification process much more simpler.
7.3 Conclusion

This thesis work has identified a suitable OWL based reasoner that could perform reasoning over RDF dataset efficiently, and also presented a tool that could verify arbitrary safety related rules over Linked data format. Thus for the research question to identify a suitable OWL reasoner, Konclude reasoner is a top performing reasoner which could perform reasoning over Linked data much efficiently. Also it is feasible to express the rules in a suitable syntax, like Manchester Syntax to verify arbitrary safety related rules over Linked data.
Bibliography


[16] OWL. url: https://www.w3.org/RDF/.


Appendix A

Grammar

grammar Condition;

/* Parser Rules */

condition : (expr+)? EOF;
expr
  : expr And expr        # andExpr
    | expr Or expr        # orExpr
    | LPar expr RPar      # parExpr
    | prop MIN Numerical expr # eqExpr
    | prop some expr       # someExpr
    | prop only expr       # onlyExpr
    | prop value dataValue valueExpr
    | id                   # idExpr
    | not id               # idExpr
;

id : Identifier;
prop:Identifier;
dataValue:Identifier;

/* Lexical Tokens */

And : 'AND';
Or : 'OR';
APPENDIX A. GRAMMAR

LPar : ' ( ';
RPar : ' ) ';
Equals : ' = ';
some : ' some ';
only : ' only ';
MIN : ' MIN ';
value : ' value ';
not : ' not ';
NEWLINE: ( ' \n ' ) { skip (); };

Numerical : [1−9] [0−9]*;
Data
  : [ true ]
  | [ false ]
  | [A]
  | [B]
  | [C]
  | [D]
  ;

// Using generic identifier tokens
// so that better warnings can be given in later // passes.
Identifier : [a−zA−Z_] [a−zA−Z0−9_] *;

// Skip parsing of whitespace
// but save on hidden channel to
// enable retrieval of original string.
WhiteSpace : [ \t\r\n ] + \rightarrow channel(HIDDEN);

// Invalid character rule is
// used so that the lexer pass never fails.
InvalidChar : .;