INTEGRITY CONSTRAINTS FOR THE SEMANTIC WEB: AN OWL 2 DL EXTENSION

By

Jiao Tao

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

Deborah L. McGuinness, Thesis Adviser

James A. Hendler, Member

Peter Fox, Member

Chris Welty, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2012
(For Graduation May 2012)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the extensive applications of Semantic Web technologies, an increasing amount of data has been published and consumed on the Semantic Web. Given some instance data, one might wonder if the data is ready for use. To answer this question, some data validation mechanisms are needed to ensure the data correctness and integrity. While syntax and semantics validation is supported by existing tools, the Semantic Web still lacks integrity constraint (IC) support. The standardized knowledge representation language, i.e., Web Ontology Language (OWL), has a standard semantics, that adopts the Open World Assumptions (OWA) and the non-Unique Name Assumptions (nUNA). This standard semantics is typically suitable for the distributed knowledge representation scenarios on the Web, where the knowledge about the domain comes from distributed sources and complete knowledge about the domain cannot be assumed. But this standard semantics makes it difficult to use OWL for closed world integrity constraint validation purposes when the knowledge about some parts of the domain is complete. This is because closed world constraint validation requires the Closed World Assumptions (CWA) and the Unique Name Assumptions (UNA) which contrast with the OWA and the nUNA respectively.

In this thesis, we propose an OWL 2 DL extension to support integrity constraints on the Semantic Web. With this approach, OWL can be used not only as a knowledge representation language for open world reasoning, but also as an integrity constraint language for closed world constraint validation. The main contributions of this thesis work are as follows:

- An integrity constraint semantics for OWL 2 DL that adopts the CWA and the weak UNA thus interpreting OWL axioms as integrity constraints.

- A sound and complete solution to integrity constraint validation by reduction to conjunctive query ($\mathrm{DCQ}^{\mathbf{not}}$) answering.

- A solution to explanation and repair of integrity constraint violations based on explanations of answers to conjunctive queries $\mathrm{DCQ}^{\mathbf{not}}$.

- A prototype implementation that we used to evaluate some relatively well published semantic web instance data from real applications: Wine, MLSO, CEDAR, BCODMO, Datagov.

This thesis work addresses an important aspect of data validation on the Semantic Web. It supports discovery and repair of defects in the instance data thus enabling quality improvement of Semantic Web data.

CHAPTER 1
Introduction and Overview

1.1 Introduction

1.1.1 Semantic Web and OWL

The Semantic Web [9] [10] is the new generation of the World Wide Web where the semantics (i.e., meaning) of information on the Web is specified such that the content on the Web can be understood and processed by machines automatically and intelligently. The Semantic Web comprises the standardized languages such as RDF, RDFS, OWL, and SPARQL, that form the Semantic Web Stack as shown in Figure 1.1.

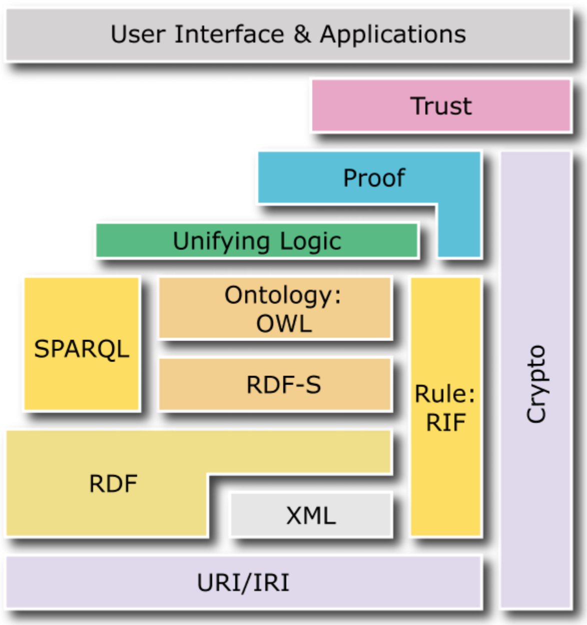

Figure 1.1: Semantic web stack

Resource Description Framework (RDF) [33] lies at the bottom of the Semantic Web stack. RDF is the language that represents information about the resources on the Web in the form of triples where each triple consists of a subject, a predicate, and an object. A set of such triples form a directed and labeled graph that is called an RDF graph in which the graph nodes represent the subjects and objects, and the edges represent the predicates. Usually RDF graphs are encoded according to RDF syntaxes such as RDF/XML [8], N3 [1], N-Triples [2], and Turtle [3]. RDF itself is not capable of representing the vocabularies such as the concepts (i.e., classes), the predicates (i.e., properties), and the relationships among them. To compensate for this, a RDF vocabulary description language, i.e., RDF Schema [12] (RDFS), is proposed, which provides means for the descriptions of classes (and class hierarchies), properties (and property hierarchies), and the domains and ranges of properties.

The language layered on top of RDF and RDFS is the Web Ontology Language (OWL 1) [61] which is an expressive ontology language that provides vocabularies for describing classes and properties as in RDFS, and adds more vocabularies for constructing classes (intersections, unions, complements, enumerations, etc.) and describing various restrictions (cardinality, somevaluesfrom, allvaluesfrom, etc.) on classes. The formal semantics of OWL 1 is based on Description Logics (DLs) [6], which is a decidable subset of First Order Logic (FOL), that is suitable for representing structured knowledge about classes, properties, and the relationships among them. There are three increasingly expressive species of OWL 1, i.e., OWL Lite, OWL DL, and OWL Full. Among the three species, OWL Lite and OWL DL correspond to DL $\mathcal{SHIF}^{(\mathcal{D})}$ and DL $\mathcal{SHOIN}^{(\mathcal{D})}$ respectively.

With the development of OWL, a new generation of OWL, i.e., OWL 2 [26], is proposed. The new features in OWL 2 include keys, property chains, qualified cardinality restrictions, expressive properties such as asymmetric, reflexive, and disjoint properties, richer data types and data ranges. OWL 2 also has two species which are OWL 2 DL and OWL 2 Full. OWL 2 DL refers to the OWL 2 ontologies interpreted using a direct semantics [43] which provides a meaning for OWL 2 based on DL $\mathcal{SROIQ}$[28]. OWL 2 Full refers to the OWL 2 ontologies interpreted using an RDF-based semantics [58] which is based on viewing OWL 2 ontologies as RDF graphs. Although from the semantics aspect the two species represent two different ways to interpret ontologies, from the syntax aspect, OWL 2 DL can be viewed as a syntactically restricted version of OWL 2 Full. Besides the two species, OWL

[1]http://www.w3.org/TeamSubmission/n3/
[2]http://www.w3.org/2001/sw/RDFCore/ntriples/
[3]http://www.w3.org/TeamSubmission/turtle/

2 also specifies three profiles [41], i.e., OWL 2 EL, OWL 2 QL, and OWL 2 RL. Each profile is a subset of OWL 2 and is suitable for certain application scenarios. Throughout the thesis, we use the terms OWL 1 and DL $\mathcal{SHOIN}^{(\mathcal{D})}$ interchangeably, OWL 2 and DL $\mathcal{SROIQ}^{(\mathcal{D})}$ interchangeably, and OWL is a general term that refers to OWL 1 and OWL 2.

Besides RDF, RDFS, and OWL, another important language in the Semantic Web stack is SPARQL. SPARQL [50] is a SQL-like query language for the Semantic Web that retrieves RDF triples from data sources. SPARQL is based on graph pattern matching, i.e., matching the graph patterns of SPARQL queries against the data sources (i.e., RDF graphs).

1.1.2 Issues in Semantic Web Data

Data validation is the process of checking data for correctness and integrity, i.e., compliance with certain standards, rules, and conventions. It is a fundamental data management problem. Without appropriate data validation, data corruption may happen which may result in failures in applications, errors in decision making, security vulnerabilities, etc. We have conducted research on Semantic Web data validation [16] [17] and have identified that the issues occurred in the validation process can be classified into three categories, i.e., syntax issues, semantics issues, and integrity issues, which are the issues related to syntax errors, logical inconsistencies, and integrity violations respectively. In this section, we will give a brief introduction to each category of issues using some examples. The given examples are composed according to the wine ontology [4].

1.1.2.1 Syntax Issues

As introduced in Section 1.1.1, data on the Semantic Web is represented as RDF triples according to RDF syntax specifications such as RDF/XML, N3, N-Triples, Turtle, etc. Given a set of Semantic Web data, syntax issues would occur if the syntax of given data is not in compliance with these RDF syntax specifications.

In Example 1, a fragment of Semantic Web data is given. In this fragment, a

[4]http://www.w3.org/TR/owl-guide/wine

"/" is missing in the last markup on line 4 (after <). Therefore this data does not conform to the RDF/XML syntax specification and a syntax issue occurs.

Example 1 *An Example of Syntax Issues*

```
<wine:Zinfandel  rdf:about="#W">
  <wine:hasMaker>
    <wine:Winery  rdf:about="#Elyse"/>
  <wine:hasMaker>
</wine:Zinfandel>
```

Syntax issues can be detected by a lot of RDF syntax validation tools. For instance, the W3C RDF validator [5] can detect the above syntax issue and give the following report

```
FatalError: The content of elements must consist of well-formed character
data or markup.[Line = 4, Column = 2].
```

1.1.2.2 Semantics Issues

Given a set of Semantic Web data, semantics issues would occur if there are logical inconsistencies in the underlying DL knowledge base of the given data.

In Example 2, individual `W` is described to have types `EarlyHarvest` and `LateHarvest`. However these two types have an `owl:disjointWith` relation in the wine ontology and should never have a common instance. Therefore there is a logical contradiction and a semantics issue occurs.

Example 2 *An Example of Semantics Issues*

```
<wine:EarlyHarvest  rdf:about="#W">
  <rdf:type>
    <owl:Class  rdf:about="#LateHarvest"/>
  </rdf:type>
</wine:EarlyHarvest>
```

[5] http://www.w3.org/RDF/Validator/

Semantics issues can be checked by a lot of OWL reasoners such as Pellet [6], Racer [7], FACT [8], Hermit [9], etc. To resolve the above semantics issue, we can either delete `W`'s membership in `EarlyHarvest` (or `LateHarvest`), or revise the wine ontology by deleting the disjoint relation between `EarlyHarvest` and `LateHarvest` when it is safe and reasonable to do so.

1.1.2.3 Integrity Issues

Integrity constraints (ICs) are the formulas that enforce the legal state of data. In databases, ICs are usually defined in terms of dependencies such as functional, inclusion, and join dependencies, and designed as a part of the database schema. ICs are heavily used in databases to ensure the data consistency and accuracy. Almost all database systems have built-in support for IC checking.

On the Semantic Web, ICs are also needed to ensure that the data in an acceptable state. Given a set of Semantic Web data, one might specify ICs using OWL (or RDFS) vocabularies. If all necessary information required by the ICs is provided in the data, then the ICs are satisfied by the data. Otherwise, integrity issues, i.e., integrity constraint violations, occur in the data.

In Example 3, individual `W` is simply declared as an instance of `Wine` without any further information. In wine ontology, `Wine` is defined as a subclass of a cardinality restriction that requires each instance of `Wine` have 1 value for property `hasMaker`. Assuming the knowledge about makers of all wines is complete and treating this cardinality restriction as an integrity constraint, an integrity issue would occur in Example 3 because a property value of `hasMaker` is missing for `W`. Similarly, if treating the other cardinality restrictions on properties `hasSugar`, `hasFlavor`, `hasBody`, `hasColor`, and `madeFromGrape` as integrity constraints, there would be more integrity issues of missing values in Example 3.

Example 3 *An Example of Integrity Issues*

$< \texttt{wine}:\texttt{Wine} \quad \texttt{rdf}:\texttt{about} = \text{"\#W"} >$

[6]http://clarkparsia.com/pellet/
[7]http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[8]http://www.cs.man.ac.uk/ horrocks/FaCT/
[9]http://hermit-reasoner.com/

1.1.3 Motivation: Lack of Integrity Constraint Supports

In last section, we have introduced three possible categories of issues in Semantic Web data. While the detection of first two categories of issues, i.e., syntax issues and semantics issues, is supported by existing tools, there have been few known solutions to the last category of issues, i.e., integrity issues. As shown in Example 3, one might want to use OWL as an IC language and to specify integrity constraints using OWL axioms such as the various cardinality restrictions on class `Wine`. However, due to the the standard semantics of OWL, these cardinality restrictions can only be used for inferences instead of integrity constraint validation: given a wine instance `W`, it is inferred that `W` has an unknown maker due to the `hasMaker` cardinality restriction; similarly, it is also inferred that `W` has some unknown values for properties `hasSugar`, `hasFlavor`, `hasBody`, `hasColor`, and `madeFromGrape`, due to the cardinality restrictions associated with these properties. Therefore, the standard semantics of OWL does not support integrity constraints.

The lack of integrity constraint supports by OWL is because the standard semantics of OWL adopts the following assumptions [10]:

- the Open World Assumptions (OWA) [11]: i.e., a statement cannot be inferred to be false on the basis of failures to prove it.
- the non-Unique Name Assumptions (nUNA): i.e., two different names may refer to the same object.

which is suitable for the distributed knowledge representation scenarios on the Semantic Web, where usually the knowledge about the domain comes from distributed sources and a complete knowledge about the domain cannot be assumed, but makes it difficult to use OWL for integrity constraint validation purposes. For instance, in Example 3, due to the OWA, it can not be inferred it is *false* that `W` has a known value for property `hasMaker` although no such known value exists. Therefore, no integrity issues would be reported even though a property value of `hasMaker` is missing for `W`.

[10]Note that these assumptions do not just pertain to OWL.

[11]OWA and Closed World Assumptions (CWA) are two opposite views on the truth values of statements. With the CWA, a statement is inferred to be false on the basis of failures to prove it.

To better describe the problem of lacking of integrity constraint supports by OWL, we use two additional examples to explain this problem.

Example 4 *Same as Example 3, suppose we have a* `Wine` *instance* `W` *without further information. One might assume the knowledge about locations of all wines is complete and use the following OWL axioms in wine ontology to express the integrity constraint that "every wine has a location":*

```
< owl : Class   rdf : ID = "Wine" >
  < rdfs : subClassOf >
    < owl : Restriction >
      < owl : onProperty   rdf : resource = "#locatedIn"/ >
      < owl : someValuesFrom   rdf : resource = "#Region"/ >
    < /owl : Restriction >
  < /rdfs : subClassOf >
< /owl : Class >
```

In this example, due to the OWA, not having a known location for wine `W` does not mean it is false that `W` has a known location. Therefore, there is no integrity issue, and we cannot use the above OWL axioms to detect (or prevent) that a wine is added to the data without the location information.

Example 5 *Suppose we have a* `Wine` *instance* `W` *that has two makers m_1 and m_2. One might use the following axioms in wine ontology to express the integrity constraint that "a wine has exactly one maker":*

```
< owl : Class   rdf : ID = "Wine" >
  < rdfs : subClassOf >
    < owl : Restriction >
      < owl : onProperty   rdf : resource = "#hasMaker"/ >
      < owl : cardinality   rdf : datatype = "#nonNegativeInteger" > 1
      < /owl : cardinality >
    < /owl : Restriction >
  < /rdfs : subClassOf >
< /owl : Class >
```

In this example, due to the nUNA, m_1 and m_2 may refer to the same object. Actually, they will be inferred to be same due to the above cardinality restriction associated with property `hasMaker`. However, here the intention to use the this cardinality restriction is not to draw this inference, but to detect an integrity constraint violation since the wine instance `W` has two makers which are not know to be same. When the information about the data are coming from multiple sources we cannot always assume that explicit inequalities about different names will be present. That is, we would like to assume that different names represent different objects by default, i.e., Unique Name Assumptions (UNA [12]), unless the names are explicitly stated to represent the same object. This requirement for the UNA for the purpose of integrity constraint validation is not compatible with the nUNA which is adopted by the standard OWL semantics.

From Example 3, 4, 5, it can be seen that the standard semantics of OWL does not support integrity constraints. The conditions trigger integrity constraint violations in database systems will generate new inferences in standard OWL-based reasoning systems. This is because the standard semantics of OWL adopts the OWA and nUNA which are opposite to the CWA and the UNA respectively that are usually called for by integrity constraint validation.

The goal of this thesis is to support integrity constraints on the Semantic Web, focusing on proposing an approach that enables OWL to be used as an integrity constraint language.

1.2 Overview

1.2.1 Contributions

In this thesis, We have presented an approach that support integrity constraints (ICs) on the Semantic Web. More precisely, We have designed and evaluated an OWL 2 DL-based approach that supports OWL to be used as an IC language.

- Proposed an alternative semantics for OWL 2 DL, i.e., the IC semantics, that adopts the CWA and the nUNA and interprets OWL axioms as ICs.

[12]With UNA, two different names always refer to two different objects.

- Designed a sound and complete solution to IC validation by reducing it to answering conjunctive queries $DCQ^{\mathbf{not}}$.
- Designed a solution to explanation and repair of IC violations.
  - Defined the general problem of explanation of conjunctive queries $DCQ^{\mathbf{not}}$ answers in OWL 2 DL.
  - Formalized the notion of justification for entailments of conjunctive queries $DCQ^{\mathbf{not}}$.
  - Reduced conjunctive query answer explanation to justification of corresponding conjunctive query entailments; Designed a set of sound and complete algorithms for the computation of conjunctive query answer explanations.
  - Described a solution to IC violation explanation based on explanations of the answers to the conjunctive queries corresponding to the ICs, and a solution to IC violation repair based on explanations.
- Implemented and evaluated this OWL 2 DL-based approach, which demonstrates its effectiveness in detecting and fixing defects in Semantic Web instance data.

1.2.2 Organization of Thesis

The thesis is organized as follows:

- In Chapter 2 we introduce the background knowledge that this thesis work is based on. This chapter discusses the Web Ontology Language (OWL), Description Logics (DLs) focusing on DL $\mathcal{SROIQ}$, epistemic extensions of DLs (EDLs), and conjunctive query answering on the Semantic Web ($DCQ^{\mathbf{not}}$).
- In Chapter 3, we review some related work, such as the work of ICs in databases, the work of IC extension to the Semantic Web which includes the approaches that enable DLs to be used as IC Languages, the approaches based on SPARQL queries, the approaches based on epistemic queries, the

approaches based on autoepistemic DLs (ADLs), and the hybrid knowledge bases (DL and LP) based approaches. We also reviewed some related work on on explanation and repair in DLs.

- In Chapter 4, 5, 6, we present an OWL 2 DL-based approach that supports ICs on the Semantic Web. First, in Chapter 4 we describe an alternative semantics for OWL 2 DL, i.e., IC semantics. We first present the formalization of this IC semantics and then explain why it enables OWL 2 DL to be used as an IC language. In Chapter 5 we describe a solution to IC validation by answering conjunctive queries $DCQ^{\mathbf{not}}$. Finally in Chapter 6 we present a solution to explanation and repair of IC violations based on explanations of answers to conjunctive queries $DCQ^{\mathbf{not}}$.

- In Chapter 7, we describe the details of an implementation of our OWL 2 DL-based approach. Then we present the evaluation results on some instance data from the real world, which shows that our approach is effective in discovering and fixing defects in the datasets. We also give a performance analysis to the system.

- Chapter 8 concludes the thesis and points out some future work.

- Finally, Appendix A, B, and C give the details for the proofs of several theorems and lemmas.

CHAPTER 2
Preliminaries

2.1 Web Ontology Language (OWL)

In 2004, the Web Ontology Language (OWL 1) [61] is released as a W3C (World Wide Consortium) recommendation. As a part of the Semantic Web stack, OWL 1 is layered on top of RDF and RDFS. OWL 1 is an expressive ontology language that provides vocabularies for describing classes and properties as in RDFS, and adds more vocabularies for constructing classes (intersections, unions, complements, enumerations, etc.) and describing various restrictions (cardinality, somevaluesfrom, allvaluesfrom, etc.) on classes. The formal semantics of OWL 1 is based on Description Logics (DLs) [6]. There are three species of OWL 1, i.e., OWL Lite, OWL DL, and OWL Full, which have increasing expressivities. Among the three species, OWL Lite is based on DL $\mathcal{SHIF}^{(\mathcal{D})}$ and OWL DL is based on DL $\mathcal{SHOIN}^{(\mathcal{D})}$.

With the development of OWL, a new generation of OWL, i.e., OWL 2 [26], is released as a W3C recommendation in 2009. The new features in OWL 2 include property chains, qualified cardinality restrictions, asymmetric properties, reflexive properties, disjoint properties, keys, richer data types and data ranges, and enhanced annotation capabilities. Similar to the three species of OWL 1, there are also two species of OWL 2 which are OWL 2 DL and OWL 2 Full. OWL 2 DL refers to the OWL 2 ontologies interpreted using a direct semantics [43] which provides a meaning for OWL 2 based on DL $\mathcal{SROIQ}$[28]. OWL 2 Full refers to the OWL 2 ontologies interpreted using an RDF-based semantics [58] which is based on viewing OWL 2 ontologies as RDF graphs and is an extension of the semantics for RDFS.

In Table 2.1, we list OWL 2 constructs and axioms, and the corresponding representations of DL $\mathcal{SROIQ}$, where $C_{(i)}$, $P_{(i)}$, $a_{(i)}$ denote OWL `Class`, `ObjectProperty`, `Individual` respectively which correspond to concept, role, object domain respectively in DLs, and n denotes a non-negative integer. To keep the presentation simple, we only focus on the object domain and not concrete domain. For a more complete introduction of concrete roles and concrete domains please refer to [43] [28]. In Table 2.1, there are three types of OWL 2 constructs and axioms: i.e., class, property, and axiom. we adopt the standard abbreviations by using $\equiv$ symbol for concept (resp. role) equivalence axioms instead of a pair of concept (resp. role) inclusion axioms, Ref(P) (resp. Irr(P)) symbol for reflexive (resp. irreflexive) role axioms, Sym(P) symbol for symmetric role axioms ($P \equiv P^-$), Asym(P) symbol for asymmetric role axioms ($P \sqsubseteq \neg P^-$), Tra(P) symbol for transitive role axioms ($PP \sqsubseteq P$).

The standard reasoning tasks in OWL include consistency checking, class satisfiability, class subsumption, instance checking.

- **Consistency Checking**: Given an OWL ontology $\mathcal{O}$, check if $\mathcal{O}$ is logically consistent, i.e., not containing any logical contradictions.
- **Class Satisfiability**: Given an OWL ontology $\mathcal{O}$ and a class C in $\mathcal{O}$, check if $C \sqsubseteq \bot$.
- **Class Subsumption**: Given an OWL ontology $\mathcal{O}$ and two classes C_1 and C_2 in $\mathcal{O}$, check if $\mathcal{O} \models C_1 \sqsubseteq C_2$.
- **Instance Checking**: Given an OWL ontology $\mathcal{O}$, a class C and an individual a in $\mathcal{O}$, check if $\mathcal{O} \models C(a)$.

In OWL DL, all of the standard reasoning problems can be reduced to the problem of ontology consistency checking: given an OWL ontology $\mathcal{O}$, classes C, C_1, C_2, and an individual a in $\mathcal{O}$, $\mathcal{O} \models C_1 \sqsubseteq C_2$ if and only if the class $C_1 \sqcap \neg C_2$ is unsatisfiable w.r.t. $\mathcal{O}$; C is satisfiable w.r.t. $\mathcal{O}$ if and only if the new OWL ontology $\mathcal{O}'$ is consistent where $\mathcal{O}' = \mathcal{O} \cup C(i)$ and i is a new individual; $\mathcal{O} \models C(a)$ if and only if the new OWL ontology $\mathcal{O}'$ is inconsistent where $\mathcal{O}' = \mathcal{O} \cup \neg C(a)$. It is known that in OWL 1 DL the consistency checking problem can be solved with sound and complete algorithms [7]; In OWL 2 DL, class satisfiability and subsumption can be reduced to the problem of determining the consistency of a class, which can also be solved with sound and complete algorithms [28].

OWL 2 specifies three profiles [41], i.e., OWL 2 EL, OWL 2 QL, and OWL 2 RL, which are three subsets of OWL 2. The profiles are defined to achieve efficient reasoning by restricting the structure of OW 2 ontologies thus sacrificing some necessary expressive power.

- **OWL 2 EL**: The motivation for OWL 2 EL is to support applications utilizing ontologies containing very large numbers of classes and/or properties. OWL 2 EL places restrictions on the language constructs that can be used: OWL 2 EL does not support universal quantifications to a class expression or a data range, cardinality restrictions, disjunction, class negation, enumerations involving more than one individual, disjoint properties, irreflexive, inverse, functional, inverse-functional, symmetric, and asymmetric object properties. The logic underpinning of OWL 2 EL is the EL family of DLs [2] [3] for which the ontology consistency, class subsumption, and instance checking can be decided in polynomial time.

- **OWL 2 QL**: OWL 2 QL is designed to support applications that utilize ontologies containing very large volumes of instance data, and for which query answering is the most important reasoning task. OWL 2 QL has quite limited expressive power. While providing the features necessary to express conceptual models, i.e., UML class diagrams and ER diagrams, OWL 2 QL does not support a lot of features such as existential and universal quantifications to a class expression or a data range, existential quantifications to an individual or a literal, self and cardinality restrictions, enumeration of individuals and literals, disjunction, property inclusions involving property chains, functional, inverse-functional, transitive properties, individual equality assertions and negative property assertions, and keys. OWL 2 QL is based on DL Lite$_R$ which is a variant of DL Lite [13]. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems which has complexity LOGSPACE w.r.t. to the size of the instance data. The other reasoning tasks such as ontology consistency and class subsumption can be solved in polynomial time.

- **OWL 2 RL**: OWL 2 RL supports the applications that call for scalable reasoning without losing too much expressive power. OWL 2 RL restricts the use of constructs to certain syntactic positions. With these restrictions, there would be no inferences of existence of unnamed individuals thus avoiding the problem of nondeterministic reasoning. The standard reasoning tasks of OWL 2 RL can be performed in polynomial time by utilizing rule-based reasoning technologies.

2.2 Description Logics (DLs)

Description Logics (DLs) [6] are a family of knowledge representation (KR) formalisms with formal and logic-based semantics. DLs are a (decidable) subset of First Order Logic (FOL). The basic syntactic building blocks in DLs are atomic concepts, atomic roles, and individuals, which represent sets of objects, relations between objects, objects respectively, and correspond to the unary predicates, binary predicates, and constants respectively in FOL.

Given the basic building blocks, complex concepts can be built from them inductively using concept constructors. DLs are distinguished by the constructors they provide. The attribute logic, DL $\mathcal{AL}$ [57], is the base language of DLs that provides constructors $\neg$, $\sqcap$, $\exists$, and $\forall$. The complex concepts in DL $\mathcal{AL}$ are constructed as follows:

$C, D \leftarrow A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top$

where A is an atomic concept, $\top$ is the universal concept which represents the universal set of all individuals in the domain, $\bot$ is the bottom concept (empty concept), R is an atomic role, C, D are concepts. Note that, in $\mathcal{AL}$, only atomic concepts are allowed to appear in scope of negation $\neg$, and only the top concept $\top$ is allowed in scope of existential quantifications $\exists$.

With the development of DLs, more expressive DLs have emerged. These expressive DLs provide additional operators including concept constructors, role constructors, etc. These operators distinguish the expressivities of different DLs. A naming convention shown in Table 2.2 is used to denote these operators and corresponding DL expressivities. The basis of expressive DLs is DL $\mathcal{ALC}$ which is $\mathcal{AL}$ with complement $\neg C$ where C is not just limited to atomic concepts. In addition to the concept constructors provided by $\mathcal{AL}$, $\mathcal{ALC}$ also allows $C \sqcup D$, i.e.,

$\neg(\neg C \sqcap \neg D)$, and $\exists R.C$, i.e., $\neg(\forall R.\neg C)$. Expressive DLs extend the expressive power of DL $\mathcal{ALC}$ by providing additional operators such as $\mathcal{S}$, $\mathcal{H}$, $\mathcal{O}$, $\mathcal{I}$, $\mathcal{N}$, $\mathcal{Q}$, $\mathcal{F}$, $\mathcal{R}$, and $\mathcal{D}$. Among these operators, $\mathcal{O}$ denotes nominals [56] which are individuals used to construct classes; $\mathcal{N}$ denotes the unqualified cardinality restriction in form of $= n.R$, $\geq n.R$, and $\leq n.R$; $\mathcal{Q}$ denotes the qualified cardinality restriction in form of $= n.R.C$, $\geq n.R.C$, and $\leq n.R.C$; $\mathcal{R}$ denotes the expressive roles including limited complex role inclusions in form of $R \circ S \sqsubseteq R$ or $S \circ R \sqsubseteq R$, role reflexivity, role irreflexivity, symmetric roles, asymmetric roles, disjoint roles, the universal role $\mathcal{U}$, and role local reflexivity $\exists R.\text{Self}$. By combining different operators together, DLs of different expressivities can be formed which then are served as the logical underpinning of different OWL ontologies. For instance, OWL Lite, OWL DL, and OWL 2 DL correspond to DL $\mathcal{SHIF}^{(\mathcal{D})}$, DL $\mathcal{SHOIN}^{(\mathcal{D})}$, DL $\mathcal{SROIQ}^{(\mathcal{D})}$ respectively. We will give a more detailed introduction to DL $\mathcal{SROIQ}^{(\mathcal{D})}$ in Section 2.2.1.

A DL knowledge base (KB) is a collection of DL axioms (logical sentence in FOL) about concepts, roles, and individuals. A typical DL KB comprises three components, a TBox, an RBox, and an ABox.

- **TBox**: A TBox consists of a set of concept inclusion axioms in form of $C \sqsubseteq D$ which states that concept C is contained in concept D.
- **RBox**: An RBox consists of a set of role inclusion axioms in form of $R \sqsubseteq S$ which states that role R is contained in role S, or other axioms about roles.
- **ABox**: An ABox consists of a set of axioms that are either concept assertions in form of $C(a)$ which states that object a belongs to concept C, or role assertions in form of $R(a, b)$ which states that objects a and b are related by role R, or other assertions.

2.2.1 Description Logic $\mathcal{SROIQ}$: Syntax and Semantics

In this section, we give a brief description of the syntax and semantics of the Description Logic $\mathcal{SROIQ}$ [28], which is the logical underpinning of OWL 2 DL [44]. More details can be found in [28]. To keep the presentation simple, we will focus only on the object domain and not data types, i.e., a simplified version of the concrete domain [4], though the approach described here can easily be extended to handle data types.

A $\mathcal{SROIQ}$ vocabulary $V = (N_C,\ N_R,\ N_I)$ is a triple where N_C, N_R, and N_I are non-empty and pair-wise disjoint sets of *atomic concepts*, *atomic roles* and *individual names* respectively. The set of $\mathcal{SROIQ}$ roles (roles, for short) is the set $N_R \cup \{R^- \mid R \in N_R\}$, where R^- denotes the inverse of the atomic role R. Concepts are defined inductively using the following grammar:

$$C \leftarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid \geq nR.C \mid \exists R.\text{Self} \mid \{a\}$$

where $A \in N_C$, $a \in N_I$, $C_{(i)}$ a concept, R a role. For expressions in the form $\geq nR.C$ where $n > 1$ we require R to be a simple role as defined in [28]. We use the following standard abbreviations for concept descriptions:

$$\begin{aligned}
&\bot = C \sqcap \neg C, \quad \top = \neg\bot, \quad C \sqcup D = \neg(\neg C \sqcap \neg D)\\
&\exists R.C = (\geq 1\, R.C), \quad \forall R.C = \neg(\exists R.\neg C)\\
&\geq n\, R = (\geq n\, R.\top), \quad \leq nR.C = \neg(\geq n+1\, R.C)\\
&\{a_1, \ldots, a_n\} = \{a_1\} \sqcup \cdots \sqcup \{a_n\}.
\end{aligned}$$

A $\mathcal{SROIQ}$-*interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ w.r.t. a vocabulary $V = (N_C,\ N_R,\ N_I)$ consists of a nonempty set $\Delta^{\mathcal{I}}$ which is the *interpretation domain*, and $.^{\mathcal{I}}$ which is the *interpretation function* that maps $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, $a \in N_I$ to an element of $\Delta^{\mathcal{I}}$. The interpretation is extended to inverse roles and complex concepts as follows:

$$(R^{-})^{\mathcal{I}} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(\geq nR.C)^{\mathcal{I}} = \{x \mid \#\{y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$$
$$(\exists R.\text{Self})^{\mathcal{I}} = \{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\}$$
$$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$$

where # denotes the cardinality of a set.

A $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$ comprises of TBox axioms, RBox axioms, and ABox axioms, as shown in Table 2.3, where C, D are concepts, $R_{(i)}$ is a role, a, b are individual names. We use the standard abbreviations for axioms by using $\equiv$ symbol for concept (resp. role) equivalence axioms instead of a pair of concept (resp. role) inclusion axioms, Tra(R) for transitive roles ($RR \sqsubseteq R$), Sym(R) for symmetric roles ($R \equiv R^{-}$), Asym(R) for asymmetric roles ($R \sqsubseteq \neg R^{-}$), Ref(R) (resp. Irr(R)) for reflexive (resp. irreflexive) roles, Dis(R_1, R_2) for disjoint roles. We say that an interpretation $\mathcal{I}$ w.r.t vocabulary V *satisfies* a $\mathcal{SROIQ}$ axiom α, denoted $\mathcal{I} \models \alpha$, if α can be constructed using the elements in V and the corresponding condition on interpretation $\mathcal{I}$ in Table 2.3 is satisfied. The interpretation $\mathcal{I}$ is a *model* of the KB $\mathcal{K}$ if it satisfies all the axioms in $\mathcal{K}$. We define $Mod(\mathcal{K})$ to be the set of all $\mathcal{SROIQ}$-interpretations that are models of $\mathcal{K}$.

The typical inferences in DL $\mathcal{SROIQ}$ include KB consistency checking, concept satisfiability, and concept subsumption, which correspond to ontology consistency checking, class satisfiability, and class subsumption in OWL.

- **KB Consistency Checking**: Given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$, if there exists no model for $\mathcal{K}$, i.e., there is no interpretation $\mathcal{I}$ that satisfies all axioms in $\mathcal{K}$, then $\mathcal{K}$ is inconsistent, otherwise $\mathcal{K}$ is consistent.

- **Concept Satisfiability**: Given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$ and a concept C, if there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$ then C is satisfiable w.r.t. $\mathcal{K}$, otherwise C is not satisfiable w.r.t. $\mathcal{K}$.

- **Concept Subsumption**: Given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$ and two concepts C_1 and C_2, if for every model $\mathcal{I}$ of $\mathcal{K}$, ${C_1}^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$, then C_2 subsumes C_1 w.r.t. $\mathcal{K}$, otherwise C_2 does not subsume C_1 w.r.t. $\mathcal{K}$.

Besides the above inferences, another typical inference in DL is axiom entailment: given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$ and an axiom α, we say $\mathcal{K}$ *entails* α, written as $\mathcal{K} \models \alpha$, if $\mathcal{I} \models \alpha$ for all models $\mathcal{I} \in Mod(\mathcal{K})$.

2.3 Epistemic Description Logics (EDLs)

As the logical foundation of the Web Ontology Language (OWL), the basic framework of Description Logics provides the means for ontology modeling. However, in some knowledge modeling scenarios, the basic framework of DLs is not enough to capture all requirements, and extensions of DLs are needed to provide additional features such as modal operators, epistemic operators, temporal operators, probability and fuzzy logic, defaults, etc. In this section, we briefly introduce the epistemic extensions of DLs (EDLs) related to this thesis work. The descriptions of other extensions can be found in [5].

Epistemic Description Logics (EDLs) [19, 54, 18] is the family of logics that are extension of DLs with the epistemic operator **K** which intuitively refers to the knowledge base *knows*.

The basic EDL is $\mathcal{ALCK}$ [19, 54, 18] which extends DL $\mathcal{ALC}$ with the operator **K**. The syntax of $\mathcal{ALCK}$ is as follows:

$C \longleftarrow \top | \bot | C_a | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \neg C | \exists R.C | \forall R.C | \mathbf{K}C$

$R \longleftarrow R_a | \mathbf{K}R_a$

Where C_a denotes an atomic concept, $C_{(i)}$ denotes a concept, R_a denotes an atomic role, and R denotes a role. Note that, the epistemic operator **K** can occur before any $\mathcal{ALC}$ concepts but only before atomic roles.

The semantics of $\mathcal{ALCK}$ is defined according to the Kripke style possible-world semantics. In the Kripke style possible-world semantics, each Kripke frame is a pair $\langle \mathcal{W}, \triangleleft \rangle$ consisting of a set of possible worlds $\mathcal{W}$ and an accessibility relation $\triangleleft$

between the worlds in $\mathcal{W}$. For $\mathcal{ALCK}$, $\mathcal{W}$ is a set of first-order interpretations $\mathcal{I}$, $\lhd$ is the universal relation on $\mathcal{W}$, i.e., $\lhd = \mathcal{W} \times \mathcal{W}$, and it is assumed that every world in $\mathcal{W}$ shares the same fixed domain which is the set of possible individuals, i.e., the Constant Domain Assumption (CDA) and the Rigid Designators Assumption (RDA). Let $\mathcal{O}$ be the set of all individuals. By the assumptions CDA and RDA, given an interpretation $\mathcal{I}$ in $\mathcal{W}$ and an individual a, we have $\Delta^{\mathcal{I}} = \mathcal{O}$ and $a^{\mathcal{I}} = a$. That is, the domain (Δ) of each interpretation ($\mathcal{I}$) is the set of all individuals ($\mathcal{O}$), and the interpretation of each individual ($a^{\mathcal{I}}$) is itself (a). Note that, the Rigid Designators Assumption forces the Unique Name Assumption (UNA): given two different individual names a and b, the interpretations of them are $a^{\mathcal{I}} = a$ and $b^{\mathcal{I}} = b$ thus are different.

An epistemic interpretation is defined as a pair $(\mathcal{I}, \mathcal{W})$ where $\mathcal{I}$ is a first-order interpretation and $\mathcal{W}$ is a set of first-order interpretations, which correspond to the real world and the possible worlds respectively in the Kripke style possible-world semantics. Given an epistemic interpretation $(\mathcal{I}, \mathcal{W})$, the interpretation function $.^{(\mathcal{I},\mathcal{W})}$ maps a concept to a subset of $\mathcal{O}$, a role to a subset of $\mathcal{O}\times\ \mathcal{O}$, an individual to an element in $\mathcal{O}$ as follows:

$\top^{\mathcal{I},\mathcal{W}} = \Delta^{\mathcal{I}} = \mathcal{O}$

$\bot^{\mathcal{I},\mathcal{W}} = \emptyset$

$(C_a)^{\mathcal{I},\mathcal{W}} = (C_a)^{\mathcal{I}}$

$(R_a)^{\mathcal{I},\mathcal{W}} = (R_a)^{\mathcal{I}}$

$(C_1 \sqcap C_2)^{\mathcal{I},\mathcal{W}} = C_1^{\mathcal{I},\mathcal{W}} \cap C_2^{\mathcal{I},\mathcal{W}}$

$(C_1 \sqcup C_2)^{\mathcal{I},\mathcal{W}} = C_1^{\mathcal{I},\mathcal{W}} \cup C_2^{\mathcal{I},\mathcal{W}}$

$(\neg C)^{\mathcal{I},\mathcal{W}} = \mathcal{O} \setminus C^{\mathcal{I},\mathcal{W}}$

$(\exists R.C)^{\mathcal{I},\mathcal{W}} = \{s \in \mathcal{O} \mid \exists t.(s,t) \in R^{\mathcal{I},\mathcal{W}} \wedge t \in C^{\mathcal{I},\mathcal{W}}\}$

$(\forall R.C)^{\mathcal{I},\mathcal{W}} = \{s \in \mathcal{O} \mid \forall t.(s,t) \in R^{\mathcal{I},\mathcal{W}} \to t \in C^{\mathcal{I},\mathcal{W}}\}$

$(\mathbf{K}C)^{\mathcal{I},\mathcal{W}} = \bigcap_{\mathcal{J}\in\mathcal{W}} C^{\mathcal{J},\mathcal{W}} = \{a \in \mathcal{O} \mid \forall \mathcal{J} \in \mathcal{W}, a \in C^{\mathcal{J},\mathcal{W}}\}$

$(\mathbf{K}R_a)^{\mathcal{I},\mathcal{W}} = \bigcap_{\mathcal{J}\in\mathcal{W}} (R_a)^{\mathcal{J},\mathcal{W}} = \{(a,b) \in \mathcal{O} \times \mathcal{O} \mid \forall \mathcal{J} \in \mathcal{W}, (a,b) \in (R_a)^{\mathcal{J},\mathcal{W}}\}$

$a^{\mathcal{I},\mathcal{W}} = a^{\mathcal{I}} = a$

It can be seen that $\mathbf{K}C$ is interpreted as the set of objects that are instances of C in every first-order interpretation in $\mathcal{W}$. That is, $\mathbf{K}C$ are the individuals that are *known* to be instances of C in $\mathcal{W}$. $\mathbf{K}R$ can be understood similarly. Therefore, the epistemic operator $\mathbf{K}$ refers to the knowledge that is *known*.

An $\mathcal{ALCK}$ knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ comprises a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, where $\mathcal{T}$ is a collection of concept inclusion axioms whose concepts are in $\mathcal{ALCK}$, and $\mathcal{A}$ is a collection of concept or role assertions whose concepts and roles are in $\mathcal{ALCK}$.

A concept inclusion axiom $C_1 \sqsubseteq C_2$, a concept assertion $C(a)$, and an role assertion $R(a, b)$, is satisfied by an epistemic interpretation $(\mathcal{I}, \mathcal{W})$ if it is true that $(C_1)^{\mathcal{I},\mathcal{W}} \subseteq (C_2)^{\mathcal{I},\mathcal{W}}$, $a \in C^{\mathcal{I},\mathcal{W}}$, $(a, b) \in R^{\mathcal{I},\mathcal{W}}$ respectively.

An epistemic model for an $\mathcal{ALCK}$ knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is a maximal non-empty set of first-order interpretations $\mathcal{W}$ such that for each $\mathcal{I} \in \mathcal{W}$, the epistemic interpretation $(\mathcal{I}, \mathcal{W})$ satisfies all of the axioms in Σ. Note that, it is assumed that an $\mathcal{ALCK}$ knowledge base Σ knows is Σ itself. That is, an $\mathcal{ALCK}$ knowledge base Σ knows nothing except the contents in Σ. To capture this notion of *minimal knowledge*, the maximality condition of an epistemic model $\mathcal{W}$ is enforced. This is because more interpretations in $\mathcal{W}$ are regarded as possible (i.e., possible worlds) if less knowledge is known by Σ: adding an interpretation to $\mathcal{W}$ may falsify a sentence that is satisfied in $\mathcal{W}$, therefore reducing the set of known knowledge.

An $\mathcal{ALCK}$ knowledge base Σ is satisfiable if there exists an epistemic model for it. Otherwise, Σ is unsatisfiable. An $\mathcal{ALCK}$ knowledge base Σ logically implies an assertion α, denoted as $\Sigma \models \alpha$, if for every epistemic model $\mathcal{W}$ of Σ, it is true that $\forall \mathcal{I} \in \mathcal{W}$, the epistemic interpretation $(\mathcal{I}, \mathcal{W})$ satisfies α. Note that, here, the logical consequence $\models$ relation is non-monotonic: given an empty $\mathcal{ALCK}$ knowledge base, i.e., $\Sigma = \emptyset$, we have $\Sigma \models (\neg\mathbf{K}C \wedge \neg\mathbf{K}\neg C)(a)$ since the knowledge base Σ does not know whether a is an instance of C or not; after adding assertion $C(a)$ to Σ we have $\Sigma \models \mathbf{K}C(a)$; that is, adding a sentence to Σ may reduce the set of consequences of Σ.

$\mathcal{ALCK}$ can be used for a lot of purposes. For instance, the epistemic operator $\mathbf{K}$ can be used in queries to form epistemic queries, or to formulate integrity constraints; $\mathcal{ALCK}$ can also be used to capture the semantics for procedure rules, etc. In the OWL 2 DL-based approach we propose, the IC semantics for OWL 2 DL is defined similarly to how the EDLs semantics is defined.

2.4 Conjunctive Query Answering

On the Semantic Web, query answering is an important service that retrieves the targeted answers from the data sources on the Web. Treating the data sources as DL knowledge bases allows us to get answers by invoking the corresponding DL inference procedures. This approach is called conjunctive query answering in DLs. In this section, we briefly introduce the this query approach.

Conjunctive query plays an important role as an expressive query language for DLs. Given a $\mathcal{SROIQ}$ KB $\mathcal{K}$ w.r.t. vocabulary $V = (N_C, N_R, N_I)$, a *conjunctive query*(CQ) Q [48] over $\mathcal{K}$ is an expression of the form

$Q(\bar{x}) \leftarrow conj(\bar{x}, \bar{y})$

where $\bar{x}$ is a tuple of *distinguished variables*, i.e., variables can be mapped to only individual names in N_I, $\bar{y}$ is a tuple of *non-distinguished variables*(not occurring in $\bar{x}$), i.e., variables can be mapped to existentially quantified individuals in $\mathcal{K}$, and $conj(\bar{x}, \bar{y})$ is a conjunction of *atomic query atoms* q of the form

$q \leftarrow C(x) \mid R(x, y) \mid \neg R(x, y) \mid x = y \mid x \neq y$

where x, y are either variables in $\bar{x}$ or $\bar{y}$ or individual names in N_I, C is a concept in N_C, and R is a role in N_R. We say that $Q(\bar{x})$ is the *query head* of Q, $conj(\bar{x}, \bar{y})$ is the *query body* of Q, and $\bar{x}$ is the *answer variable* of Q.

A significant amount of research has been devoted to study the algorithms and computational complexities of answering CQ in DLs of different expressivities [24] [23] [38] [25]. It turns out even in the DLs less expressive than $\mathcal{SROIQ}$, i.e., $\mathcal{SHIQ}$, the complexity is exponential in the size of the KB and double exponential in the size of the query [24], and the complexity becomes even higher for more expressive DLs. Considering that in realistic scenarios most frequently occurred queries do not contain non-distinguished variables, a category of queries, i.e., *distinguished conjunctive query*(DCQ), is proposed and studied [59] [64]. A DCQ query $Q(\bar{x})$ is a conjunction of query atoms, i.e., $conj(\bar{x}, \bar{y})$, where all query variables, i.e., $\bar{x}$ and $\bar{y}$, are distinguished. Note that, different from normal CQ queries where all distinguished variables are treated as answer variables, in DCQ queries, only some query variables, i.e., $\bar{x}$, are put in query head $Q(\bar{x})$ and are treated as are answer variables. The evaluations [59] show that DCQ query answering can be optimized via a lot of mechanisms thus can be answered more efficiently even over KBs with large ABoxes without compromising the soundness and completeness.

A recent work [64] extends the capability of DCQ with negative inference retrieval and names this category of queries as *distinguished conjunctive queries with negation as failure* (DCQ$^{\mathbf{not}}$). The syntax of DCQ$^{\mathbf{not}}$ is defined as follows:

$$Q \leftarrow q \mid Q_1 \wedge Q_2 \mid \mathbf{not}\, Q$$

where a DCQ$^{\mathbf{not}}$ query could be a single atomic query atom, a conjunction of DCQ$^{\mathbf{not}}$ queries, or a negation of a DCQ$^{\mathbf{not}}$ query. The semantics of DCQ$^{\mathbf{not}}$ queries is given in terms of DL interpretations defined in Section 2.2.1. Given a $\mathcal{SROIQ}$ KB $\mathcal{K}$ and a DCQ$^{\mathbf{not}}$ query $Q(\bar{x}) \leftarrow conj(\bar{x}, \bar{y})$, we define an *assignment* $\sigma : \{\bar{x} \cup \bar{y}\} \rightarrow N_I$ to be a mapping from the variables used in the query to individual names in $\mathcal{K}$. We define $\sigma(Q)$ to denote the *application* of an assignment σ to a query Q such that variables in Q are replaced with individuals according to the mapping. We say a $\mathcal{SROIQ}$ KB $\mathcal{K}$ *entails* a DCQ$^{\mathbf{not}}$ query Q w.r.t. the assignment σ, or the *query entailment* $\mathcal{K} \models^{\sigma} Q$ holds, if:

$$\begin{array}{lll} \mathcal{K} \models^{\sigma} q & \text{iff} & \mathcal{K} \models \sigma(q) \\ \mathcal{K} \models^{\sigma} Q_1 \wedge Q_2 & \text{iff} & \mathcal{K} \models^{\sigma} Q_1 \text{ and } \mathcal{K} \models^{\sigma} Q_2 \\ \mathcal{K} \models^{\sigma} \mathbf{not}\, Q & \text{iff } \nexists \sigma' \text{ s.t.} & \mathcal{K} \models^{\sigma'} \sigma(Q) \end{array}$$

We define the *answers to a query*, $\mathbf{Ans}(Q(\bar{x}), \mathcal{K})$, to be the set of tuples of individuals, such that for each tuple, there exist some assignments σ that map answer variables $\bar{x}$ to this tuple and $\mathcal{K}$ entails query $Q(\bar{x})$ w.r.t. σ. That is, $\mathbf{Ans}(Q(\bar{x}), \mathcal{K}) = \{\bar{a} \mid \exists \sigma, \text{ s.t. } \bar{x} \rightarrow \bar{a} \in \sigma, \mathcal{K} \models^{\sigma} Q\}$. We say that a query Q is *true* w.r.t. a KB $\mathcal{K}$, denoted $\mathcal{K} \models Q$, if there is at least one answer for the query, and *false* otherwise.

Note that using only distinguished variables in DCQ$^{\mathbf{not}}$ queries changes the query answers as illustrated by the following example.

Example 6 *Suppose we have the following KB and query*

$\mathcal{K} = \{A \equiv \exists R.\top, A(a), A(b), R(b,c)\}$

$Q(x) \leftarrow A(x) \wedge R(x,y)$.

Then, there is only one assignment $\sigma : x \rightarrow b, y \rightarrow c$ for which $\mathcal{K} \models^{\sigma} Q$ holds. Therefore $\mathbf{Ans}(Q(x), \mathcal{K})$ *contains a single answer (b). Note that (a) is not an answer to $Q(x)$ even though $R(a,y)$ is satisfied in all of the interpretations of the KB. This is due to the fact that all variables in $Q(x)$ are distinguished and need to be mapped to individual names.*

We use the abbreviation $Q_1 \vee Q_2$ for **not** (**not** $Q_1 \wedge$ **not** Q_2). It is easy to see that $\mathcal{K} \models^{\sigma} Q_1 \vee Q_2$ iff $\mathcal{K} \models^{\sigma} Q_1$ or $\mathcal{K} \models^{\sigma} Q_2$. Note that this abbreviation is not same as conventional disjunctive queries and interacts with the disjunctions in KBs in a restricted way as the next example shows.

Example 7 *Suppose we have the following KB and query*

$\mathcal{K} = \{A \sqsubseteq B \sqcup C, A(a)\}$

$Q(x) \leftarrow B(x) \vee C(x)$.

Then, $\mathbf{Ans}(Q, \mathcal{K})$ *is empty. This is because the answers to queries $B(x)$ and $C(x)$ are both empty and their union is empty even though a is known to be an instance of $B \sqcup C$.*

The negation as failure **not** in DCQ$^{\mathbf{not}}$ queries could be at non-atomic level. That is, **not** could be used not just before an atomic query atom but also before any legal DCQ$^{\mathbf{not}}$ queries as shown by the following example.

Example 8 *Suppose we have the following KB and query*

$\mathcal{K} = \{C \sqsubseteq A, \exists s.\top \sqsubseteq A, D \sqsubseteq B, p(a_1, b), A(a_1), A(a_2), C(a_2), s(a_3, b)\}$

$Q(x) \leftarrow A(x) \wedge$ ***not*** $(p(x,y) \wedge B(y))$.

Then, the answers to query $Q(x)$ is $\mathbf{Ans}(Q(x), \mathcal{K}) = \{(a_1), (a_2), (a_3)\}$.

Note that, this negative inference retrieval at non-atomic level increases the expressivity of DCQ queries without incurring additional complexity for query answering: to answer the above query $Q(x)$ we just need to simply retrieve all instances of A that are not answers to the sub-query $p(x,y) \wedge B(y)$.

Besides the conjunctive query answering approachs in DLs, SPARQL is another important query approach. SPARQL [47] is a W3C recommended RDF query language that has been widely used on the Semantic Web. SPARQL query answering treats the data sources as RDF graphs allows us to get answers by matching the SPARQL graph patterns against the RDF graphs. It can be seen that it is very different from conjunctive query answering in DLs which views query answering as a process that is based on inferences on the underlying DL knowledge bases of the data sources. However, there are still some connections between the two categories of query answering: first, SPARQL allows querying over OWL ontologies via OWL entailment regimes [13]; second, SPARQL incorporates the *NOT EXISTS* filter to support negation as failure [14] which is also supported by DCQ$^{\mathbf{not}}$ queries; third, it is known that SPARQL has the same expressive power as non-recursive Datalog programs [15] [1] and can express DCQ$^{\mathbf{not}}$ queries. Therefore, SPARQL query engine can be used to execute $DCQ^{\mathbf{not}}$ queries.

[13]http://www.w3.org/TR/rdf-sparql-query/#sparqlBGPExtend

[14]http://www.w3.org/TR/sparql11-query/#negation

[15]The detailed relationship and translation between SPARQL and Datalog has been investigated in [49].

Type	OWL 2 Construct and Axiom	DL $\mathcal{SROIQ}$ Representation
Class	`ObjectIntersectionOf(`$\mathtt{C_1 \ldots C_n}$`)`	$C_1 \sqcap \ldots \sqcap C_n$
	`ObjectUnionOf(`$\mathtt{C_1 \ldots C_n}$`)`	$C_1 \sqcup \ldots \sqcup C_n$
	`ObjectComplementOf(C)`	$\neg C$
	`ObjectOneOf(`$\mathtt{a_1 \ldots a_n}$`)`	$\{a_1\} \sqcup \ldots \sqcup \{a_n\}$
	`ObjectAllValuesFrom(P C)`	$\forall P.C$
	`ObjectSomeValuesFrom(P C)`	$\exists P.C$
	`ObjectHasValue(P a)`	$\exists P.\{a\}$
	`ObjectHasSelf(P)`	$\exists P.\mathtt{Self}$
	`ObjectExactCardinality(n P)`	$= n.P$
	`ObjectExactCardinality(n P C)`	$= n.P.C$
	`ObjectMaxCardinality(n P)`	$\leq n.P$
	`ObjectMaxCardinality(n P C)`	$\leq n.P.C$
	`ObjectMinCardinality(n P)`	$\geq n.P$
	`ObjectMinCardinality(n P C)`	$\geq n.P.C$
Property	`ObjectInverseOf(P)`	P^-
Axiom	`SubClassOf(`$\mathtt{C_1\ C_2}$`)`	$C_1 \sqsubseteq C_2$
	`EquivalentClasses(`$\mathtt{C_1 \ldots C_n}$`)`	$C_1 \equiv \ldots \equiv C_n$
	`DisjointClasses(`$\mathtt{C_1\ C_2}$`)`	$C_1 \sqsubseteq \neg C_2$
	`DisjointClasses(`$\mathtt{C_1 \ldots C_n}$`)`	$C_i \sqsubseteq \neg C_j$, $(1 \leq i \leq n,\ 1 \leq j \leq n$, s.t. $i \neq j)$
	`DisjointUnionOf(CN `$\mathtt{C_1 \ldots C_n}$`)`	$CN = (C_1 \sqcup \ldots \sqcup C_n)$ $C_i \sqsubseteq \neg C_j$ $(1 \leq i \leq n,\ 1 \leq j \leq n$, s.t. $i \neq j)$
	`SubObjectPropertyOf(`$\mathtt{P_1\ P_2}$`)`	$P_1 \sqsubseteq P_2$
	`SubObjectPropertyOf( ObjectPropertyChain(`$\mathtt{P_1 \ldots P_n}$`) P)`	$P_1 \ldots P_n \sqsubseteq P$
	`ObjectPropertyDomain(P C)`	$\exists P.\top \sqsubseteq C$
	`ObjectPropertyRange(P C)`	$\exists P^-.\top \sqsubseteq C$
	`EquivalentObjectProperties(`$\mathtt{P_1 \ldots P_n}$`)`	$P_1 \equiv \ldots \equiv P_n$
	`DisjointObjectProperties(`$\mathtt{P_1\ P_2}$`)`	$P_1 \sqsubseteq \neg P_2$
	`DisjointObjectProperties(`$\mathtt{P_1 \ldots P_n}$`)`	$P_i \sqsubseteq \neg P_j$, $(1 \leq i \leq n,\ 1 \leq j \leq n$, s.t. $i \neq j)$
	`InverseObjectProperties(`$\mathtt{P_1\ P_2}$`)`	$P_1 \equiv {P_2}^-$
	`FunctionalObjectProperty(P)`	$\top \sqsubseteq \leq 1.P$
	`InverseFunctionalObjectProperty(P)`	$\top \sqsubseteq \leq 1.P^-$
	`ReflexiveObjectProperty(P)`	Ref(P)
	`IrreflexiveObjectProperty(P)`	Irr(P)
	`SymmetricObjectProperty(P)`	Sym(P)
	`AsymmetricObjectProperty(P)`	Asym(P)
	`TransitiveObjectProperty(P)`	Tra(P)
	`SameIndividual(`$\mathtt{a_1 \ldots a_n}$`)`	$\{a_1\} \equiv \ldots \equiv \{a_n\}$
	`DifferentIndividuals(`$\mathtt{a_1\ a_2}$`)`	$\{a_1\} \sqsubseteq \neg\{a_2\}$
	`DifferentIndividuals(`$\mathtt{a_1 \ldots a_n}$`)`	$\{a_i\} \sqsubseteq \neg\{a_j\}$, $(1 \leq i \leq n,\ 1 \leq j \leq n$, s.t. $i \neq j)$
	`ClassAssertion(C a)`	$C(a)$
	`ObjectPropertyAsssertion(P `$\mathtt{a_1\ a_2}$`)`	$P(a_1, a_2)$
	`NegativeObjectPropertyAssertion(P `$\mathtt{a_1\ a_2}$`)`	$\neg P(a_1, a_2)$

Table 2.1: Correspondence between OWL 2 and DL $\mathcal{SROIQ}$

Operator	**DL Expressivity**
$\mathcal{AL}$	Attribute Logic
$\mathcal{ALC}$	Attribute Logic + Complement
$\mathcal{S}$	$\mathcal{ALC}$ + Transitive Roles
$\mathcal{H}$	Role Hierarchy
$\mathcal{O}$	Nominals
$\mathcal{I}$	Inverse Roles
$\mathcal{N}$	Unqualified Cardinality Restrictions
$\mathcal{Q}$	Qualified Cardinality Restrictions
$\mathcal{F}$	Functional Roles
$\mathcal{R}$	Expressive Roles
$\mathcal{D}$	Data Types

Table 2.2: Naming convention of DLs

Type	**Axiom**	**Condition on $\mathcal{I}$**
TBox	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RBox	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
	$R_1 \ldots R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
	Tra(R)	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
	Sym(R)	$\forall x, y \in \Delta : \langle x, y \rangle \in R^{\mathcal{I}}$ implies $\langle y, x \rangle \in R^{\mathcal{I}}$
	Asym(R)	$\forall x, y \in \Delta : \langle x, y \rangle \in R^{\mathcal{I}}$ implies $\langle y, x \rangle \notin R^{\mathcal{I}}$
	Ref(R)	$\forall x \in \Delta : \langle x, x \rangle \in R^{\mathcal{I}}$
	Irr(R)	$\forall x \in \Delta : \langle x, x \rangle \notin R^{\mathcal{I}}$
	Dis(R_1, R_2)	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} = \emptyset$
ABox	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
	$R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
	$\neg R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}$
	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

Table 2.3: Axiom satisfactions in $\mathcal{SROIQ}$-interpretation $\mathcal{I}$

CHAPTER 3
Related Work

In this chapter, we review some related work. First, in Section 3.1, we discuss some work of representing and validating ICs in databases. Then, in Section 3.2, we discuss several approaches supports ICs on the Semantic Web, focusing on the extension to DLs. Finally, in Section 3.3, we discuss some related work on explanation and repair in DLs;

3.1 ICs in Databases

Integrity Constraints (ICs) originate in databases such as relational databases and deductive databases. They are used to represent and enforce the legal states of databases, where the databases can be viewed as knowledge bases (KBs) that are represented as a set of first order logic (FOL) sentences.

There are different views on ICs in databases. One conventional view on ICs is they are FOL sentences [46, 51, 37]. Along this view, there are two approaches to define IC satisfactions of KBs. One approach defines IC satisfaction as KB consistencies [34]. That is, given a KB and an IC, the KB satisfies this IC iff the KB augmented with the IC is consistent. Another approach defines IC satisfaction as KB entailments [51, 37]. That is, given a KB and an IC, the KB satisfies this IC iff the KB entails (FOL entailment) the IC, i.e., the IC is true in all models of the KB.

Reiter argued against [52, 53] the above conventional view on ICs and the two consistency-based and entailment-based IC satisfaction definitions. Reiter presented the view that ICs are epistemic in nature and are about "what the knowledge base knows". According to the epistemic view, ICs should be epistemic FOL sentences that incorporates the epistemic operator **K** into normal FOL sentences, where **K** denotes what a knowledge base *knows*. Nicolas et. al. [45] investigated IC representations in relational databases by which the typical ICs are modeled as a set of dependencies also using epistemic FOL sentences. With this epistemic view, checking if a FOL KB Σ satisfies an IC α is to examine if the KB Σ entails (epistemic entailment) α. Let $\mathcal{W}$ be the set of models of Σ, if $\forall \mathcal{I} \in \mathcal{W}$, α is true in $\mathcal{I}$ and $\mathcal{W}$, then the IC α is satisfied by Σ, otherwise it is violated.

Another view on ICs is from Lloyd and Topor [36]. He treats ICs in deductive databases as regular FOL queries. Given a deductive database KB $\mathcal{K}$ and an IC α, checking if $\mathcal{K}$ satisfies α is querying $\mathcal{K}$ with the FOL query corresponding to α. Deductive databases cannot answer arbitrary FOL queries so Lloyd-Topor provides a set of transformation rules (which is now famously known as the Lloyd-Topor transformation) to convert a FOL query to a general logic program that may include negation as failure (NAF). Such a logic program can be executed over deductive databases in a straight-forward and efficient manner yielding a practical approach for validation of ICs expressed as FOL formula. If the given deductive KB $\mathcal{K}$ entails (non-monotonic entailment) the logic program corresponding to the IC α, then the IC α is satisfied by $\mathcal{K}$, otherwise it is violated. The Lloyd-Topor approach can be illustrated using Example 4. First, the IC "every wine should have a location" can be expressed as the following FOL sentence

$\texttt{Wine}(x) \to \exists y.(\texttt{locatedIn}(x, y) \wedge \texttt{Region}(y))$.

Then, a rule can be constructed whose head is empty (denoted by the special symbol $\perp$) and whose body is the negation of the above FOL sentence representing the IC

$\perp \text{ :- } \textbf{not}\,(\texttt{Wine}(x) \to \exists y.(\texttt{locatedIn}(x, y) \wedge \texttt{Region}(y)))$

which states that if the negation of the IC (the FOL sentence) can be proven to be true from the deductive database then an empty head is inferred that indicates an violation of the given IC. Using the Lloyd-Topor transformation rules, the above rule can be first translated into the following logic program

$\perp \text{ :- } \texttt{Wine}(x) \wedge \textbf{not}\,(\exists y.(\texttt{locatedIn}(x, y) \wedge \texttt{Region}(y)))$,

then translated into the following two rules

$\perp \text{ :- } \texttt{Wine}(x) \wedge \textbf{not}\,\texttt{P}(x, y)$

$\texttt{P}(x, y) \text{ :- } \texttt{locatedIn}(x, y) \wedge \texttt{Region}(y)$

which constitute a general logic program, where $\texttt{P}$ is a new predicate generated by the transformation. IC satisfaction then can be checked by executing this logic program over the given deductive database where $\perp$ indicates an IC violation.

Among the different related work of ICs in databases, the epistemic view on ICs and the Lloyd-Topor transformation has important influence on our work. First, we agree with the epistemic view on ICs and defined an IC semantics for OWL that also embodies this epistemic view; Second, in our solution to IC validation, the translation rules from OWL IC axioms to $DCQ^{\mathbf{not}}$ queries is similar to how the Lloyd-Topor transformation rules are defined.

3.2 IC Extension to the Semantic Web

There are different approaches that support ICs on the Semantic Web. In this section, we discuss several representative categories of approaches that either directly aiming at enabling DLs to be used as IC languages, or extend the Semantic Web languages (especially DLs) with other formalisms thus indirectly supporting ICs. We first give a detailed introduction to the direct approaches in Section 3.2.1, then briefly look at four indirect approaches in Section 3.2.2, Section 3.2.3, Section 3.2.4, and Section 3.2.5.

3.2.1 Approaches Based on Enabling DLs to be Used as IC Languages

The closest related work to ours is the approaches that directly aim at enabling DLs as IC languages. These approaches reuse the syntax of DLs and devise alternative semantics for DLs to support ICs. One notable effort along this line is a proposal by Motik et al. [42] which is based on a minimal Herbrand model semantics of DLs: given a DL KB $\mathcal{K}$ consisting of a TBox, an ABox, and an integrity constraint TBox, an axiom in the integrity constraint TBox is satisfied by $\mathcal{K}$ if all minimal Herbrand models of $\mathcal{K}$ satisfy this IC axiom.

However, as shown in the following cases, this approach may result in counterintuitive results or a significant modeling burden.

First, unnamed individuals can satisfy ICs, which is not desirable for closed world data validation.

Example 9 *Consider a DL KB $\mathcal{K}$ that contains a wine instance w and its unknown location, and an IC axiom α that every wine should have a known location:*

$\mathcal{K} = \{A \equiv \exists \mathtt{locatedIn}.\mathtt{Region}, \mathtt{Wine}(w), A(w)\}$

$\alpha : \texttt{Wine} \sqsubseteq A$

According to this approach, since w has a location in every minimal Herbrand model of $\mathcal{K}$, the IC axiom α is satisfied by $\mathcal{K}$, even though the location of w is unknown.

Second, if an integrity constraint needs to be satisfied only by named individuals, then a special concept O has to be added into the original IC axiom, and every named individual should be asserted as an instance of O. This adds a significant maintenance burden, in particular when modification to the original TBox or ABox is not allowed or too costly. Further, the next example shows that with this approach the intuition behind the IC may not be correctly captured.

Example 10 *Suppose we have a DL KB $\mathcal{K}$ where there are two possible locations for a wine w and an IC axiom α:*

$$\mathcal{K} = \{B \equiv \exists\texttt{locatedIn}.\{r_1, r_2\}, \texttt{Wine}(w), B(w), \texttt{Region}(r_1), \texttt{Region}(r_2), O(w), O(r_1), O(r_2)\}$$

$$\alpha : \texttt{Wine} \sqsubseteq \exists\texttt{locatedIn}.(\texttt{Region} \sqcap O)$$

The intuition behind the IC axiom α is that the location of every wine should be known. Even though we do not know the location of w is r_1 or r_2, α is still satisfied by $\mathcal{K}$ according to this approach because in every minimal Herbrand model of $\mathcal{K}$, w has a location that is also an instance of `Region` and O.

Third, disjunctions and ICs may interact in unexpected ways.

Example 11 *Consider the following DL KB $\mathcal{K}$ where there are two categories for wine and an IC axiom α defined on one of the categories:*

$$\mathcal{K} = \{\texttt{Wine} \sqsubseteq \texttt{Category}_1 \sqcup \texttt{Category}_2, \texttt{Wine}(w)\}$$

$$\alpha : \texttt{Category}_1 \sqsubseteq \exists\texttt{categoryType}.\top$$

Since we are not sure that w belongs to $\texttt{Category}_1$, it is reasonable to assume that the integrity constraint will not apply to w and it will not be violated. However, α is violated according to this approach because there is a minimal Herbrand model where w belongs to $\texttt{Category}_1$ but it does not have a `categoryType` value.

Another effort along this line is a proposal by [60] which is based on translation from OWL IC axioms to SPARQL queries. This approach is similar to our work from the aspect of translation-based IC validation approach but there are some important differences: first, this approach only describes an approach for IC validation without providing an IC semantics for OWL; second, the translation in this approach is from OWL IC axioms to SPARQL queries while our work use the translation from OWL IC axioms to $DCQ^{\mathbf{not}}$ queries; third, this approach and our approach have different implementations: the former is based on the SPARQL query engine in the OWL reasoner Pellet, the latter is based on DLV system.

3.2.2 Approaches Based on SPARQL Queries

Besides the direct approaches for ICs, there are several indirect approaches for ICs. One category of approaches is based on SPARQL queries. In this line of work one proposal is [**?**] which identifies several categories of integrity issues that are frequently encountered in the Semantic Web data, detects each category of integrity issues using SPARQL queries. Three important categories of integrity issues described in this proposal are: (i) integrity issues related to an individual type including Unexpected Individual Type (UIT) issues, Redundant Individual Type (RIT) issues, Non-specific Individual Type (NSIT) issues; (ii) integrity issues related to a property value including Missing Property Value (MPV) issues, Excessive Property Value (EPV) issues. This approach has several limitations: first, it does not provide an semantics for ICs; second, it only describes several categories of integrity issues without a comprehensive description for all integrity issues in the data.

3.2.3 Approaches Based on Epistemic Queries

Similar to the work on ICs in databases that we have introduced in Section 3.1 which view ICs as epistemic FOL sentences, there are also some approaches that extend DLs or FOLs with epistemic queries, therefore support ICs.

The epistemic queries could be written using the epistemic language $\mathcal{ALCK}$ [54, 18, 5]. As we have introduced in Section 2.3, $\mathcal{ALCK}$ is an extension of DL $\mathcal{ALC}$ with the epistemic operator $\mathbf{K}$, where $\mathbf{K}$ intuitively refers to what a knowledge base knows. In $\mathcal{ALCK}$, an epistemic concept $\mathbf{K}C$ (C is an $\mathcal{ALC}$ concept) represents the individuals that are *known* to be instances of concept C, and an epistemic role $\mathbf{K}R_a$ (R_a is an $\mathcal{ALC}$ atomic role) represents the pair-wise individuals that are *known* to be associated by role R_a. Therefore, using the epistemic operator $\mathbf{K}$ in queries allows us to retrieve what a knowledge base knows. According to this approach, given an ordinary DL $\mathcal{ALC}$ knowledge base Φ that contains only ABox axioms, we can use the epistemic $\mathcal{ALC}$ concepts as the query language to formulate ICs. For example, the IC "every person should be male or female" could be represented using the following epistemic concept disjunction

$$\neg\mathbf{K}\texttt{Person} \sqcup (\mathbf{K}\texttt{Male} \sqcup \mathbf{K}\texttt{Female})$$

which denotes the individuals that either are not known to be persons or are known to be male or female. It can be seen that the KB Φ satisfies the IC if and only if the instance retrieval of the above epistemic concept disjunction returns every individual name occurring in Φ. It is known that querying DL $\mathcal{ALC}$ knowledge bases that contain only ABox axioms with $\mathcal{ALC}$ concept query language has complexity PSpace-complete. And adding $\mathbf{K}$ into $\mathcal{ALC}$ concept query language does not increase the query complexity. Therefore, answering epistemic queries that are formulated with $\mathcal{ALCK}$ concepts over DL $\mathcal{ALC}$ knowledge bases that contain only ABox axioms also has PSpace-complete complexity.

The epistemic queries can also be written using more expressive epistemic languages such as EQL-Lite [14]. EQL-Lite is an epistemic FOL query language that incorporates the epistemic operator $\mathbf{K}$ into FOL queries which can be posted to standard FOL knowledge bases. Since every OWL axiom corresponds to a FOL formula we can represent an OWL IC axiom using EQL-Lite queries. For instance, the IC "every wine should have an location" in Example 4 of Section 1.1.3 can be formulated as the follow query $Q(x)$:

$$Q(x) \leftarrow \mathbf{K}\texttt{Wine}(x) \wedge \exists y.(\mathbf{K}\texttt{locatedIn}(x, y) \wedge \mathbf{K}\texttt{Region}(y))$$

which returns all individuals in the KB that satisfy the IC. Although the data complexity of answering EQL-Lite queries in DL Lite is LOGSPACE, it would require substantially more effort to support EQL-Lite in more expressive DLs and the complexity results are still unknown.

3.2.4 Approaches Based on ADLs

Another category of approaches to extend DLs with ICs is based on the Autoepistemic Description Logics (ADLs) [20, 55, 21]. ADLs are the extensions of DLs with two modal operators **K** and **A**, that can be used in DL concepts and roles and intuitively refer to what a knowledge base knows and the assumptions respectively. The basic ADLs is ADL $\mathcal{ALCK}_{\mathcal{NF}}$ which extends $\mathcal{ALC}$ with operators **K** and **A**.

With this approach, the ICs can be represented as $\mathcal{ALCK}_{\mathcal{NF}}$ axioms. For example, the IC "every wine should have an location" in Example 4 of Section 1.1.3 can be represented using the following $\mathcal{ALCK}_{\mathcal{NF}}$ axiom

$\mathbf{K}\texttt{Wine} \sqsubseteq \exists\mathbf{A}\texttt{locatedIn}.\mathbf{A}\texttt{Region}$

which states that each *known* instance of wine is *assumed* to have a location. According to this approach, given an ordinary DL $\mathcal{ALC}$ knowledge base Φ that contains only ABox axioms, adding the above $\mathcal{ALCK}_{\mathcal{NF}}$ axiom to Φ, if the new KB, i.e., an ADL KB, is satisfiable, then the above IC is satisfied, otherwise the IC is violated.

One recent work along this line is a proposal [63] that characterizes five categories of integrity issues in OWL DL data, i.e., UIT, RIT, NSIT, MPV, EPV, and provides a SPARQL query-based solution to the detection of these integrity issues, which is proved to be sound.

Note that, both the epistemic queries-based approaches described in Section 3.2.3 and the ADLs-based approaches described in this section are based on epistemic extensions of DLs thus consistent with the epistemic nature of ICs. But they are two different categories of approaches. First, they represent ICs differently. The former approaches formulate ICs as $\mathcal{ALCK}$ epistemic queries *posted to* standard DL KBs while the latter approaches represent ICs as $\mathcal{ALCK}_{\mathcal{NF}}$ epistemic axioms *embedded into* DL KBs to form ADL KBs; Second, they validate ICs differently. The former approaches validate ICs by checking the answers to the $\mathcal{ALCK}$ epistemic queries, while the latter approaches decide IC satisfactions by checking the satisfiability of ADL KBs.

3.2.5 Approaches Based on Hybrid KBs (DL+LP)

Another line of indirect approaches to extend DLs with ICs is based on the hybrid knowledge bases that comprise DL KBs and logic programs (LP). There has been significant amount of research [22, 29, 40] has been devoted to combine DLs with logic programming, which is aimed at overcoming the shortcomings of the basic framework of DLs such as the incapability of performing non-monotonic or closed-world reasoning.

With this line of approaches, ICs can be expressed as rules and are coupled with DL KBs. For instance, according to the proposal [22], ICs are represented as rules without heads. The IC in Example 4 is expressed with rules as follows:

$\bot \leftarrow DL[\texttt{Wine}](x), \mathbf{not}\,\texttt{P}(x, y)$

$\texttt{P}(x, y) \leftarrow DL[\texttt{locatedIn}](x, y), DL[\texttt{Region}](y)$

where **not** is the negation as failure, atoms with the prefix DL are *DL atoms*, and all the other atoms are *non-DL atoms*. Among the non-DL atoms, $\bot$ is a special predicate representing an empty rule head. The addition of ICs in form of above rules to a DL KB constitutes a hybrid KB. The DL atoms are evaluated as queries to the DL KB, and the non-DL atoms are evaluated as normal. According to these approaches, the detection of IC violations is reduced to checking if the special predicate $\bot$ is entailed by the hybrid KB.

3.3 Explanation in DLs

The research on explanation of DL reasoning has been going on for years. The earliest work is a proposal by McGuinness et al. [39] where a proof-oriented approach for explanation of the general subsumption reasoning in CLASSICAL has been proposed. Later on, the research on explanation in DLs has been conducted in more directions, such as incoherent terminologies, inconsistent KBs, unsatisfiable concepts, axiom entailments, query answers, etc. The work on latter two directions is more related to our work.

In the line of work on explanation of axiom entailments, one notable effort is by Kalyanpur et. al. [32]. This work formalized the notion of justifications of axiom entailments in OWL DL, which are minimal sets of axioms in the KB that are sufficient to produce an entailment. In this work, some algorithms are provided to compute a single justification for an entailment, and a hitting set tree (HST)-based approach is described to compute all justifications based on this single justification. In addition to the work on finding justifications at the axiom level, Kalyanpur et. al. [31] also investigated on finding justifications at a finer granularity level, i.e., identifying parts of the axioms relevant for an axiom entailment. Another work on fine-grained justifications for axiom entailments is a proposal by Horridge et. al. [27] which defined two categories of fine-grained justifications, i.e., laconic justifications and precise justifications, and provided an algorithm to compute these justifications.

In the line of work on explanation of query answers in DLs, one notable work is by Borgida et. al. [11] which discussed the problem of explaining answers to queries over ABoxes in the setting of DL lite. This work identified two sub-problems of query answer explanation, i.e., explanation of positive answers which explains why certain tuple of individuals is an answer to the given query, and explanation of failed answers which explains why certain tuple of individuals is not an answer to the given query. For positive answer explanation, they presented a strategy to find the explanations. For failed answer explanation, they only briefly discussed a naive and heuristic idea without giving more details.

The several work discussed above is related to our OWL 2 DL-based approach for ICs on the Semantic Web in several aspects: first, we formally define the problem of conjunctive query answer explanation in expressive DLs (DL $\mathcal{SROIQ}$) and describe a solution to it that addresses not only positive answer explanation but also negative answer explanation; second, we define the notion of justification for conjunctive query entailments, which is inspired by the notion of axiom entailment justification; third, in our solution to IC violation repair, the repairs are computed based on explanations. Note that, although the notion of hitting set is used both in the proposal [32] and our work, it is used in very different ways: in the former work, a hitting set tree (HST)-based algorithm is used to compute all justifications; while in our work, MHSs are used to compute repairs based on positive justifications.

CHAPTER 4
An IC Semantics for OWL 2 DL

As illustrated by the the motivating examples, i.e., Example 4 and 5 in Section 1.1.3, we observe a strong need to use OWL as an IC language. That is, we would like to adopt the OWA and the nUNA for parts of the domain where knowledge is incomplete, and to use the CWA with the UNA otherwise. This calls for the ability to combine the open world reasoning with closed world constraint validation.

In Chapter 4, 5, and 6, we are going to present an OWL 2 DL-based approach [64] [62] we propose for ICs on the Semantic Web: (1) we first propose an alternative IC semantics for OWL 2 DL, which enables developers to augment OWL ontologies with IC axioms. Standard OWL axioms in the ontologies are used to compute inferences with the standard open world semantics and ICs are used to validate instance data with the closed world semantics; (2) we show that IC axioms can be translated to conjunctive queries according to the given translation rules, therefore IC validation can be reduced to conjunctive query answering; (3) then, we address the general problem of explanation of conjunctive query answers in OWL 2 DL by an approach of justifying corresponding conjunctive query entailments. We show that explanation of IC violations can be reduced to explaining answers to conjunctive queries, and repair of IC violations can be computed based on the explanations.

As described in Chapter 3, Reiter [52, 53] stated that ICs are epistemic in nature and are about "what the knowledge base knows". He said that ICs should be epistemic FOL queries that will be asked against a standard KB that does not contain epistemic axioms. We agree with Reiter's assessment of the epistemic nature of ICs and believe this is an appropriate semantics for ICs. We will now describe an alternative IC semantics for OWL 2 DL (DL $\mathcal{SROIQ}$), which is similar to how the semantics of epistemic DL $\mathcal{ALCK}$ [18] is defined in Section 2.3. Then, we discuss how our IC semantics addresses the issues discussed in Section 1.1.3 and how it enables OWL to be used as an IC language.

4.1 Formalization

A $\mathcal{SROIQ}$ vocabulary $V = (N_C,\ N_R,\ N_I)$ is a triple where N_C, N_R, and N_I are non-empty and pair-wise disjoint sets of atomic concepts, atomic roles, and individual names respectively. We define an *IC-interpretation* $\mathcal{I},\mathcal{U} = (\Delta^{\mathcal{I}},\ \cdot^{\mathcal{I},\mathcal{U}})$ w.r.t a vocabulary V where $\mathcal{I} = (\Delta^{\mathcal{I}},\ \cdot^{\mathcal{I}})$ is a $\mathcal{SROIQ}$ interpretation w.r.t V and $\mathcal{U} = \mathcal{I}_1$, ..., $\mathcal{I}_n$ is a set where each $\mathcal{I}_k = (\Delta^{\mathcal{I}_k},\ \cdot^{\mathcal{I}_k})$ is a $\mathcal{SROIQ}$ interpretation w.r.t. V. The IC-interpretation function $\cdot^{\mathcal{I},\mathcal{U}}$ maps $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, $a \in N_I$ to an element of $\Delta^{\mathcal{I}}$ as follows:

$$A^{\mathcal{I},\mathcal{U}} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, x^{\mathcal{J}} \in A^{\mathcal{J}}\}$$
$$R^{\mathcal{I},\mathcal{U}} = \{\langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \mid x, y \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, \langle x^{\mathcal{J}}, y^{\mathcal{J}}\rangle \in R^{\mathcal{J}}\}$$
$$a^{\mathcal{I},\mathcal{U}} = a^{\mathcal{I}}$$

According to the above definition, $A^{\mathcal{I},\mathcal{U}}$ is the set of objects in $\Delta^{\mathcal{I}}$ that are the mappings of individual names which are instances of A in every $\mathcal{SROIQ}$ interpretation in $\mathcal{U}$. That is, $A^{\mathcal{I},\mathcal{U}}$ represents the interpretation of individual names that are *known* to be instances of A in $\mathcal{U}$. $R^{\mathcal{I},\mathcal{U}}$ can be understood similarly.

IC-interpretation $\mathcal{I},\mathcal{U}$ is extended to inverse roles and complex concepts as follows:

$$(R^-)^{\mathcal{I},\mathcal{U}} = \{\langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \mid \langle y^{\mathcal{I}}, x^{\mathcal{I}}\rangle \in R^{\mathcal{I},\mathcal{U}}\}$$
$$(C \sqcap D)^{\mathcal{I},\mathcal{U}} = C^{\mathcal{I},\mathcal{U}} \cap D^{\mathcal{I},\mathcal{U}}$$
$$(\neg C)^{\mathcal{I},\mathcal{U}} = (N_I)^{\mathcal{I}} \setminus C^{\mathcal{I},\mathcal{U}}$$
$$(\geq nR.C)^{\mathcal{I},\mathcal{U}} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \#\{y^{\mathcal{I}} \mid \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R^{\mathcal{I},\mathcal{U}} \text{ and } y^{\mathcal{I}} \in C^{\mathcal{I},\mathcal{U}}\} \geq n\}$$
$$(\leq nR.C)^{\mathcal{I},\mathcal{U}} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \#\{y^{\mathcal{I}} \mid \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R^{\mathcal{I},\mathcal{U}} \text{ and } y^{\mathcal{I}} \in C^{\mathcal{I},\mathcal{U}}\} \leq n\}$$
$$(\exists R.\text{Self})^{\mathcal{I},\mathcal{U}} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \langle x^{\mathcal{I}}, x^{\mathcal{I}}\rangle \in R^{\mathcal{I},\mathcal{U}}\}$$
$$\{a\}^{\mathcal{I},\mathcal{U}} = \{a^{\mathcal{I}}\}.$$

where $(N_I)^{\mathcal{I}} = \{x^{\mathcal{I}} \mid x \in N_I\}$ is the interpretation of N_I.

It can be seen that, an IC-interpretation $\mathcal{I},\mathcal{U}$ and a $\mathcal{SROIQ}$ interpretation $\mathcal{I}$ interpret $\neg C$ differently: a $\mathcal{SROIQ}$ interpretation $\mathcal{I}$ of $\neg C$ is the set of objects in $\Delta^{\mathcal{I}}$ that are not instances of C in $\mathcal{I}$, while an IC-interpretation $\mathcal{I}$,$\mathcal{U}$ of $\neg C$ is the set of objects in $\Delta^{\mathcal{I}}$, that are mappings of individual names (N_I) which are not instances of C in every interpretation in $\mathcal{U}$.

Note that, although the IC interpretations have some similarities to the epistemic interpretations of $\mathcal{ALCK}$ [18], there are some important differences. First, the IC interpretations are applicable to any DL $\mathcal{SROIQ}$ KBs while the epistemic interpretations in [18] are only applicable to DL $\mathcal{ALC}$ KBs. Second, in $\mathcal{ALCK}$, the epistemic interpretations adopt the Rigid Designators Assumption (RDA) thus enforce the strict UNA, which is not the case in IC interpretations.

In our IC semantics, we want to adopt the UNA, to support closed world constraint validation, but in a weak form, such that it is compatible with the nUNA adopted by the standard semantics of OWL. That is, two individual with different names are assumed to be different by default unless their equality is required for the sake of the satisfiability of the KBs. This weak UNA notion is similar to the minimal model semantics where the equality relations are treated as congruence relations and minimized.

We formalize this notion of weak UNA by defining Minimal Equality (ME) models. We start by defining the *less than relation w.r.t. equality*, denoted as $\prec_=$, between interpretations. Formally, given a $\mathcal{SROIQ}$ KB $\mathcal{K}$ and two $\mathcal{SROIQ}$ interpretations $\mathcal{I}$ and $\mathcal{J}$ of $\mathcal{K}$, we say $\mathcal{J} \prec_= \mathcal{I}$ if all of the following conditions hold:

- For every atomic concept C, $\mathcal{J} \models C(a)$ iff $\mathcal{I} \models C(a)$;
- For every atomic role R, $\mathcal{J} \models R(a,b)$ iff $\mathcal{I} \models R(a,b)$;
- $E_{\mathcal{J}} \subset E_{\mathcal{I}}$

where $E_{\mathcal{I}}$ denotes the set of equality relations between individual names (equality relations, for short) satisfied by interpretation $\mathcal{I}$:

$$E_{\mathcal{I}} = \{\langle a,b\rangle \mid a,b \in N_I \text{ s.t. } \mathcal{I} \models a = b\}$$

Here, a and b could be any individual names in $\mathcal{K}$.

Type	**Axiom**	**Condition on $\mathcal{I}$,$\mathcal{U}$**
TBox	$C \sqsubseteq D$	$C^{\mathcal{I},\mathcal{U}} \subseteq D^{\mathcal{I},\mathcal{U}}$
RBox	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I},\mathcal{U}} \subseteq R_2^{\mathcal{I},\mathcal{U}}$
	$R_1 \dots R_n \sqsubseteq R$	$R_1^{\mathcal{I},\mathcal{U}} \circ \dots \circ R_n^{\mathcal{I},\mathcal{U}} \subseteq R^{\mathcal{I},\mathcal{U}}$
	$\texttt{Ref}(R)$	$\forall x \in N_I : \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$
	$\texttt{Irr}(R)$	$\forall x \in N_I : \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$
	$\texttt{Dis}(R_1, R_2)$	$R_1^{\mathcal{I},\mathcal{U}} \cap R_2^{\mathcal{I},\mathcal{U}} = \emptyset$
ABox	$C(a)$	$a^{\mathcal{I},\mathcal{U}} \in C^{\mathcal{I},\mathcal{U}}$
	$R(a,b)$	$\langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$
	$\neg R(a,b)$	$\langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$
	$a = b$	$a^{\mathcal{I},\mathcal{U}} = b^{\mathcal{I},\mathcal{U}}$
	$a \neq b$	$a^{\mathcal{I},\mathcal{U}} \neq b^{\mathcal{I},\mathcal{U}}$

Table 4.1: Axiom satisfactions in IC-interpretation $\mathcal{I}$,$\mathcal{U}$

The Minimal Equality (ME) models, i.e., $Mod_{ME}(\mathcal{K})$, are the models of $\mathcal{K}$ with minimal equality relations between individual names. Formally, we define

$$Mod_{ME}(\mathcal{K}) = \{\mathcal{I} \in Mod(\mathcal{K}) \mid \nexists \mathcal{J}, \mathcal{J} \in Mod(\mathcal{K}), \mathcal{J} \prec_{=} \mathcal{I}\}$$

It can be seen that two different individual names are interpreted as equivalent in a ME model $\mathcal{I}$ only if this equality is necessary to make $\mathcal{I}$ being a model of $\mathcal{K}$. For example, suppose we have an axiom $a = \{b\} \sqcup \{c\}$ in $\mathcal{K}$. Then, $\forall \mathcal{I} \in Mod(\mathcal{K})$, one of the following three conditions hold: (1) $a^{\mathcal{I}} = b^{\mathcal{I}}, a^{\mathcal{I}} \neq c^{\mathcal{I}}$; (2) $a^{\mathcal{I}} = c^{\mathcal{I}}, a^{\mathcal{I}} \neq b^{\mathcal{I}}$; (3) $a^{\mathcal{I}} = b^{\mathcal{I}} = c^{\mathcal{I}}$. If (1) or (2) holds, then $\mathcal{I} \in Mod_{ME}(\mathcal{K})$, because a has to be interpreted to be equivalent to at least one of b and c to make $\mathcal{I}$ being a model of $\mathcal{K}$. Whereas for case (3), $\mathcal{I} \notin Mod_{ME}(\mathcal{K})$ since the equality relations between individual names are not minimal.

The satisfaction of an axiom α in an IC-interpretation $\mathcal{I}$,$\mathcal{U}$, denoted as $\mathcal{I}, \mathcal{U} \models \alpha$, is defined in Table 4.1, where C, D are concepts, $R_{(i)}$ is a role, a, b are individual names. Note that, when N_I is an empty set, every role is both reflexive and irreflexive, which is different from the conventional interpretations.

Given a $\mathcal{SROIQ}$ KB $\mathcal{K}$ and a $\mathcal{SROIQ}$ axiom α, the IC-satisfaction of α by $\mathcal{K}$, denoted by $\mathcal{K} \models_{IC} \alpha$, is defined as:

$$\mathcal{K} \models_{IC} \alpha \text{ iff } \forall \mathcal{I} \in \mathcal{U}, \mathcal{I}, \mathcal{U} \models \alpha, \text{where } \mathcal{U} = Mod_{ME}(\mathcal{K})$$

The above IC-satisfaction has a closed world flavor: (1) given an atomic concept C and an individual name a, if $\mathcal{K} \not\models_{IC} C(a)$ then we conclude $\mathcal{K} \models_{IC} \neg C(a)$; (2) given an role R and two individual names a and b, if $\mathcal{K} \not\models_{IC} R(a,b)$ then we conclude $\mathcal{K} \models_{IC} \neg R(a,b)$. The first conclusion holds because, if $\mathcal{K} \not\models_{IC} C(a)$ then $\exists \mathcal{I} \in \mathcal{U}$ such that $a^{\mathcal{I},\mathcal{U}}$ $(=a^{\mathcal{I}}) \notin C^{\mathcal{I},\mathcal{U}}$. That is, $\exists \mathcal{I} \in \mathcal{U}$ such that $a^{\mathcal{I}} \notin C^{\mathcal{I}}$. Therefore, $\forall \mathcal{J} \in \mathcal{U}, a^{\mathcal{J}} \notin C^{\mathcal{J},\mathcal{U}}$, thus $\forall \mathcal{J} \in \mathcal{U}, a^{\mathcal{J},\mathcal{U}} = a^{\mathcal{J}} \in (N_I)^{\mathcal{J}} \setminus C^{\mathcal{J},\mathcal{U}} = (\neg C)^{\mathcal{J},\mathcal{U}}$. That is, $\forall \mathcal{J} \in \mathcal{U}$, $a^{\mathcal{J},\mathcal{U}} \in (\neg C)^{\mathcal{J},\mathcal{U}}$. So we conclude that $\mathcal{K} \models_{IC} \neg C(a)$. The second conclusion can be proved similarly. The two conclusions show that the IC semantics of OWL has a closed world flavor: it uses the CWA for atomic concept assertions and role assertions; failure to satisfy an atomic concept assertion (resp. an role assertion) results in the satisfaction of its negation, which is not the case with the standard OWL semantics.

We define an *extended KB* as a pair $\langle \mathcal{K}, \mathcal{C} \rangle$ where $\mathcal{K}$ is a $\mathcal{SROIQ}$ KB interpreted with the standard semantics and $\mathcal{C}$ is a set of $\mathcal{SROIQ}$ axioms interpreted with the IC semantics. We say that the extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ is valid if $\forall \alpha \in \mathcal{C}, \mathcal{K} \models_{IC} \alpha$, otherwise there is an IC violation.

4.2 Discussion

It is easy to verify that the IC semantics proposed in Section 4.1 provides expected results for examples in Section 1.1.3 and Section 3.2.1, and enables OWL 2 DL to be an IC language. In Example 4, given the instance data Wine(W) and the IC axiom Wine $\sqsubseteq \exists$locatedIn.Region, we get an IC violation since the IC interpretation of Wine contains W but the IC interpretation of ($\exists$locatedIn.Region) is empty. We also get an IC violation for Example 5 because, due to the weak UNA, m_1 and m_2 are interpreted as different individuals, causing the IC interpretation of $((=1)$hasMaker$.\top)$ to be empty. In Example 9 and Example 10 there are also IC violations because the constraint requires the same named location to exist in every ME model of the KB which is not the case. Since our IC semantics is targeted at individual names, one does not need to use the concept O as in Example 10. In Example 11, there is a model of the KB where w is not an instance of Category1, which makes the IC interpretation of Category1 to be empty, therefore the constraint does not apply to w and there is no IC violation.

The following example shows how the weak UNA allows the individuals that are not explicitly asserted equal to be treated different for integrity constraint validation purposes.

Example 12 *Consider the KB $\mathcal{K}$ and the integrity constraint α:*

$$\mathcal{K} = \{C(c), R(c, d_1), R(c, d_2), D(d_1), D(d_2)\}$$
$$\alpha : C \sqsubseteq \geq 2R.D$$

With the weak UNA, d_1 and d_2 are interpreted to be different in every ME model. Therefore, the IC-interpretation of $(\geq 2R.D)$ includes c, and the integrity constraint α is satisfied by $\mathcal{K}$.

Now we illustrate another point regarding disjunctions in integrity constraints.

Example 13 *Suppose we have the KB $\mathcal{K}$ and constraint α:*

$$\mathcal{K} = \{C(a), (C_1 \sqcup C_2)(a)\}$$
$$\alpha : C \sqsubseteq C_1 \sqcup C_2$$

The integrity constraint α should be read as "every instance of C should be either a known instance of C_1 or a known instance of C_2". Since we do not know *for sure* whether a belongs to C_1 or C_2, α is expected to be violated by $\mathcal{K}$. Indeed, according to our semantics we get $C^{\mathcal{I},\mathcal{U}} = \{a^{\mathcal{I}}\}$ and $(C_1 \sqcup C_2)^{\mathcal{I},\mathcal{U}} = \emptyset$. Therefore $C^{\mathcal{I},\mathcal{U}} \not\subseteq (C_1 \sqcup C_2)^{\mathcal{I},\mathcal{U}}$ and we conclude there is an IC violation.

If we want to represent the alternative constraint: "every instance of C should be a known instance of C_1 or C_2", we can define a new name C' in the KB to substitute $C_1 \sqcup C_2$, thus having the new KB $\mathcal{K}'$ and integrity constraint α' as follows:

$$\mathcal{K}' = \{C(a), (C_1 \sqcup C_2)(a), C' \equiv C_1 \sqcup C_2\}$$
$$\alpha' : C \sqsubseteq C'$$

There is no IC violation in this version because now the disjunction is interpreted as standard OWL axioms. As the above examples show, we can model the integrity constraints to express different disjunctions in a flexible way.

CHAPTER 5
IC Validation

We have defined in Section 4.1 that, the extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ is valid if every IC axiom in $\mathcal{C}$ is IC-satisfied by $\mathcal{K}$. In this chapter, we describe how to do IC validation by first translating integrity constraint axioms to conjunctive queries with negation as failure then executing the conjunctive queries over $\mathcal{K}$. We start by describing the translation rules from IC axioms to DCQ$^{\mathbf{not}}$ queries, then provide a theorem showing that IC validation can be reduced to answering DCQ$^{\mathbf{not}}$ queries under certain conditions.

5.1 Translation Rules: from ICs to DCQ$^{\mathbf{not}}$ Queries

We now present the translation rules from IC axioms to DCQ$^{\mathbf{not}}$ queries. The translation rules are similar in the spirit to the Lloyd-Topor transformation [35]. The idea behind the translation is to translate an IC axiom into a query such that when the IC is violated the KB entails the query. In other words, whenever the answer of the query is not empty, we can conclude that the IC is violated.

The translation contains two operators: $\mathcal{T}_c$ for translating concepts and $\mathcal{T}$ for translating axioms. $\mathcal{T}_c$ is a function that takes a concept expression and a variable as input and returns a DCQ$^{\textbf{not}}$ query as the result:

$$\mathcal{T}_c(C_a, x) := C_a(x)$$

$$\mathcal{T}_c(\neg C, x) := \textbf{not}\,\mathcal{T}_c(C, x)$$

$$\mathcal{T}_c(C_1 \sqcap C_2, x) := \mathcal{T}_c(C_1, x) \wedge \mathcal{T}_c(C_2, x)$$

$$\mathcal{T}_c(\geq nR.C, x) := \bigwedge_{1 \leq i \leq n} (R(x, y_i) \wedge \mathcal{T}_c(C, y_i)) \bigwedge_{1 \leq i < j \leq n} \textbf{not}\,(y_i = y_j)$$

$$\mathcal{T}_c(\leq nR.C, x) := \textbf{not}\,(\mathcal{T}_c(\geq (n+1)R.C, x))$$

$$\mathcal{T}_c(\exists R.C, x) := \mathcal{T}_c(\geq 1R.C, x) = R(x, y) \wedge \mathcal{T}_c(C, y)$$

$$\mathcal{T}_c(\forall R.C, x) := \textbf{not}\,(R(x, y) \wedge \textbf{not}\,\mathcal{T}_c(C, y))$$

$$\mathcal{T}_c(\exists R.\mathrm{Self}, x) := R(x, x)$$

$$\mathcal{T}_c(\{a\}, x) := (x = a)$$

where C_a is an atomic concept, $C_{(i)}$ is a concept, R is a role, a is an individual, x is an input variable, and $y_{(i)}$ is a fresh variable.

$\mathcal{T}$ is a function that maps a $\mathcal{SROIQ}$ axiom to a DCQ$^{\textbf{not}}$ query as follows:

$$\mathcal{T}(C_1 \sqsubseteq C_2) := \mathcal{T}_c(C_1, x) \wedge \textbf{not}\,\mathcal{T}_c(C_2, x)$$

$$\mathcal{T}(R_1 \sqsubseteq R_2) := R_1(x, y) \wedge \textbf{not}\,R_2(x, y)$$

$$\mathcal{T}(R_1 \ldots R_n \sqsubseteq R) := R_1(x, y_1) \wedge \ldots R_n(y_{n-1}, y_n) \wedge \textbf{not}\,R(x, y_n)$$

$$\mathcal{T}(\texttt{Ref}(R)) := \textbf{not}\,R(x, x)$$

$$\mathcal{T}(\texttt{Irr}(R)) := R(x, x)$$

$$\mathcal{T}(\texttt{Dis}(R_1, R_2)) := R_1(x, y) \wedge R_2(x, y)$$

$$\mathcal{T}(C(a)) := \textbf{not}\,\mathcal{T}_c(C, a)$$

$$\mathcal{T}(R(a, b)) := \textbf{not}\,R(a, b)$$

$$\mathcal{T}(\neg R(a, b)) := R(a, b)$$

$$\mathcal{T}(a = b) := \textbf{not}\,(a = b)$$

$$\mathcal{T}(a \neq b) := (a = b)$$

where $C_{(i)}$ is a concept, $R_{(i)}$ is a role, x, $y_{(i)}$ is a variable, a and b are individuals.

Example 14 *Given the IC axiom* $\texttt{Wine} \sqsubseteq \exists\texttt{locatedIn.Region}$ *in Example 4, applying the above translation rules* $\mathcal{T}$*, we get:*

$$\begin{aligned}
&\mathcal{T}(\texttt{Wine} \sqsubseteq \exists\texttt{locatedIn.Region})\\
&:= \mathcal{T}_c(\texttt{Wine}, x) \wedge \textbf{not}\ \mathcal{T}_c(\exists\texttt{locatedIn.Region}, x)\\
&:= \texttt{Wine}(x) \wedge \textbf{not}\ (\texttt{locatedIn}(x, y) \wedge \mathcal{T}_c(\texttt{Region}, y))\\
&:= \texttt{Wine}(x) \wedge \textbf{not}\ (\texttt{locatedIn}(x, y) \wedge \texttt{Region}(y))
\end{aligned}$$

5.2 Reducing IC Validation to Answering DCQ$^{\textbf{not}}$ Queries

We have discussed in Section 5.1, we want to reduce the problem of IC validation to query answering. However, a careful examination of the problem reveals that when both the KB and the ICs use full expressivity of $\mathcal{SROIQ}$, we cannot use the query translation approach in a straightforward way.

Example 15 *Suppose we have a KB* $\mathcal{K}$ *and an integrity constraint* α*:*

$$\begin{aligned}
&\mathcal{K} = \{D(d), R(d, a), R(d, b), R(d, c), \{a\} \sqsubseteq \{b, c\}\}\\
&\alpha : D \sqsubseteq\ \leq 2R.\top
\end{aligned}$$

In all models of $\mathcal{K}$, a is either interpreted to be equivalent to b or c. Therefore, in all models of $\mathcal{K}$, d has at most two R values and $\mathcal{K}$ satisfies the integrity constraint α. However, the query translation will yield atoms in the form **not** $(y_i = y_j)$ which will be true for any individual pair in this KB. As a result, the answer for this query will include d, which incorrectly indicates an IC violation. This is due to the problematic interactions between disjunctive (in)equality axioms in $\mathcal{K}$ and cardinality restrictions in ICs: the axiom $(\{a\} \sqsubseteq \{b, c\})$ asserts a disjunctive equality between individuals, therefore the IC axiom (α) containing cardinality restrictions is satisfied in different ways in different interpretations.

To avoid such problematic interactions, we ca either prohibit cardinality restrictions in $\mathcal{SROIQ}$ ICs or prohibit disjunctive (in)equality axioms in the $\mathcal{SROIQ}$ KB. On one hand, by excluding cardinality restrictions from $\mathcal{SROIQ}$ ICs, we get $\mathcal{SROI}$ ICs. On the other hand, in $\mathcal{SROIQ}$, there are only three ways to infer (in)equality between individuals: using (1) explicit (in)equality axioms; (2) nominals (as seen above); and (3) cardinality restrictions. Obviously, explicit ABox assertions cannot be disjunctive so they are not problematic. Therefore, by excluding nominals and cardinality restrictions from the $\mathcal{SROIQ}$ KB, we get the $\mathcal{SRI}$ KB. In other words, when the ICs are in $\mathcal{SROI}$ or the KB is in $\mathcal{SRI}$, there would be no problematic interactions.

In what follows, we first give several lemmas, then present the main theorem which shows that IC validation via query answering is sound and complete for the cases where the expressivity of the extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ is either $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$. Since answers to the conjunctive queries, that correspond to IC axioms $\mathcal{C}$, are affected by the inferences computed by standard OWL axioms in $\mathcal{K}$, open world reasoning and closed world constraint validation are combined on the extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$.

Lemma 1 *Let $\mathcal{K}$ be a $\mathcal{SROIQ}$ KB, suppose $\mathcal{I} \in Mod(\mathcal{K})$ and $\mathcal{I} \notin Mod_{ME}(\mathcal{K})$, then there is a model $\mathcal{J}$ such that $\mathcal{J} \in Mod_{ME}(\mathcal{K})$ and $\mathcal{J} \prec_= \mathcal{I}$*

Proof By the definition of $Mod_{ME}(\mathcal{K})$

$$Mod_{ME}(\mathcal{K}) = \{\mathcal{I} \in Mod(\mathcal{K}) \mid \nexists \mathcal{J}, \mathcal{J} \in Mod(\mathcal{K}), \mathcal{J} \prec_= \mathcal{I}\}$$

we know $\exists \mathcal{W}_1 \in Mod(\mathcal{K})$ such that $\mathcal{W}_1 \prec_= \mathcal{I}$. If $\mathcal{W}_1 \in Mod_{ME}(\mathcal{K})$ then we find the model $\mathcal{J} = \mathcal{W}_1$, otherwise $\exists \mathcal{W}_2 \in Mod(\mathcal{K})$ such that $\mathcal{W}_2 \prec_= \mathcal{W}_1$. We continue this process. After finite (n) steps the process will terminate. We find n models $\mathcal{W}_i \in Mod(\mathcal{K})(1 \leq i \leq n), \mathcal{W}_n \prec_= \mathcal{W}_{n-1} \ldots \prec_= \ldots \mathcal{W}_2 \prec_= \mathcal{W}_1 \prec_= \mathcal{I}$ and we could not find another model $\mathcal{W}_{n+1}$ such that $\mathcal{W}_{n+1} \in Mod(\mathcal{K}), \mathcal{W}_{n+1} \prec_= \mathcal{W}_n$. This process terminates after at most n steps because $\mathcal{K}$ contains finite individual names and there are at most n models $\mathcal{W}_i$ to satisfy the $\prec_=$ relation. If one of the n models belongs to $Mod_{ME}(\mathcal{K})$ then we find the model $\mathcal{J}$, otherwise none of the n models belongs to $Mod_{ME}(\mathcal{K})$. Then by the definition of $Mod_{ME}(\mathcal{K})$, $\exists \mathcal{W}_{n+1} \in Mod(\mathcal{K}), \mathcal{W}_{n+1} \prec_= \mathcal{W}_n$, which leads to a contradiction.$\Box$

Lemma 2 *Let $\mathcal{K}$ be a $\mathcal{SROIQ}$ KB, $C \in N_C$, $R \in N_R$, and $a, b \in N_I$. Then,*
(1)$\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ iff $\forall \mathcal{J} \in Mod(\mathcal{K})$, $a^{\mathcal{J}} \in C^{\mathcal{J}}$.

(2)$\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$ *iff* $\forall \mathcal{J} \in Mod(\mathcal{K})$, $\langle a^{\mathcal{J}}, b^{\mathcal{J}} \rangle \in R^{\mathcal{J}}$.

(3)$\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $a^{\mathcal{I}} = b^{\mathcal{I}}$ *iff* $\forall \mathcal{J} \in Mod(\mathcal{K})$, $a^{\mathcal{J}} = b^{\mathcal{J}}$.

Proof We only show the proof for the first case. The proofs for case (2) and (3) are similar.

$\Rightarrow$ Assume to the contrary that $\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $\exists \mathcal{J} \in Mod(\mathcal{K})$, $a^{\mathcal{J}} \notin C^{\mathcal{J}}$ (or $\mathcal{J} \not\models C(a)$). This means $\mathcal{J} \notin Mod_{ME}(\mathcal{K})$. By Lemma 1 we know $\exists \mathcal{W} \in Mod_{ME}(\mathcal{K})$ such that $\mathcal{W} \prec_{=} \mathcal{J}$. Since $\mathcal{W} \in Mod_{ME}(\mathcal{K})$, we have $a^{\mathcal{W}} \in C^{\mathcal{W}}$ which means $\mathcal{W} \models C(a)$. This is a contradiction because, it is known that $\mathcal{W} \prec_{=} \mathcal{J}$, by the definition of $\prec_{=}$, $\mathcal{W} \models C(a)$ iff $\mathcal{J} \models C(a)$, which is not true.

$\Leftarrow$ This is trivially true since $Mod_{ME}(\mathcal{K}) \subseteq Mod(\mathcal{K})$.□

Lemma 3 *Let* $\mathcal{K}$ *be a* $\mathcal{SRI}$ *KB,* $\forall a, b \in N_I$. *Then.*

(1) either $\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $a^{\mathcal{I}} = b^{\mathcal{I}}$.

(2) or $\forall \mathcal{I} \in Mod_{ME}(\mathcal{K})$, $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Proof Assume to the contrary of the lemma that $\exists \mathcal{J} \in Mod_{ME}(\mathcal{K})$ for which $a^{\mathcal{J}} = b^{\mathcal{J}}$ and $\exists \mathcal{W} \in Mod_{ME}(\mathcal{K})$ for which $a^{\mathcal{W}} \neq b^{\mathcal{W}}$. It is easy to see that $\mathcal{K} \not\models a = b$ and $\mathcal{K} \not\models a \neq b$. We construct a new interpretation $\mathcal{J}$' from $\mathcal{J}$ as follows:

- $c^{\mathcal{J}'} = c^{\mathcal{J}}$ for all $c \in N_I$ where $c \neq a$.
- $a^{\mathcal{J}'} = s$ where s is a fresh new value that is different from $c^{\mathcal{J}'}$ where $c \in N_I \backslash \{a\}$.
- For every concept C, if $a^{\mathcal{J}} \in C^{\mathcal{J}}$ then $C^{\mathcal{J}'} = C^{\mathcal{J}} \cup a^{\mathcal{J}'}$, otherwise $C^{\mathcal{J}'} = C^{\mathcal{J}}$.
- For every role R, if $\langle a^{\mathcal{J}}, m^{\mathcal{J}} \rangle \in R^{\mathcal{J}}$ then $R^{\mathcal{J}'} = R^{\mathcal{J}} \cup \langle a^{\mathcal{J}'}, m^{\mathcal{J}'} \rangle$; if $\langle m^{\mathcal{J}}, a^{\mathcal{J}} \rangle \in R^{\mathcal{J}}$ then $R^{\mathcal{J}'} = R^{\mathcal{J}} \cup \langle m^{\mathcal{J}'}, a^{\mathcal{J}'} \rangle$; otherwise $R^{\mathcal{J}'} = R^{\mathcal{J}}$.

We can see $\mathcal{J}' \in Mod(\mathcal{K})$. This is because changing the interpretation of individual a to s in $\mathcal{J}$' will only affect interpretation of nominals, cardinality restrictions ($\geq nR.C$ with $n > 1$), explicit equality ($=$) and inequality ($\neq$) axioms. First, we know the $\mathcal{SRI}$ KB is free of nominals and cardinality restrictions in form of $\geq nR.C$ with $n > 1$ (it still allows $\geq 1R.C$ which is equivalent to $\exists R.C$); Second, there is neither the explicit equivalence axiom $a = b$ nor the explicit inequivalence axiom $a \neq b$ in

$\mathcal{K}$ because $\mathcal{K} \not\models a = b$ and $\mathcal{K} \not\models a \neq b$. So we conclude that $\mathcal{J}$' satisfies all axioms in $\mathcal{K}$. That is, $\mathcal{J}' \in Mod(\mathcal{K})$. It can be seen that for every atomic concept C in $\mathcal{K}$, $\mathcal{J}' \models C(w)$ iff $\mathcal{J} \models C(w)$, and for every atomic role R in $\mathcal{K}$, $\mathcal{J}' \models R(u, v)$ iff $\mathcal{J} \models R(u, v)$, where w, u, v are individual names in N_I. It can also be seen $E_{\mathcal{J}'} \subset E_{\mathcal{J}}$ because $\mathcal{J}$' and $\mathcal{J}$ satisfy the same set of individual equality relations except $a = b$, for which only $\mathcal{J}$ satisfies it. According to the definition of $\prec_=$ we have $\mathcal{J}' \prec_= \mathcal{J}$. Furthermore, by the definition of $Mod_{ME}(\mathcal{K})$, we know $\mathcal{J} \notin Mod_{ME}(\mathcal{K})$, which is a contradiction of our assumption.□

Lemma 4 *Given an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ with expressivity $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or expressivity $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$, we say that $\mathcal{K} \models_{IC} C(a)$ iff $\mathcal{K} \models \mathcal{T}_c(C, a)$, where C is a concept in IC axioms $\mathcal{C}$ and $a \in N_I$.*

Proof Sketch The proof is constructed inductively on the structure of concept C which is a $\mathcal{SROIQ}$ concept. The complete proof is presented in Appendix A.□

We now obtain the main theorem of this chapter which shows that, given an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$, when the expressivity of the KB is either within $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or within $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$, $\mathcal{K}$ IC-satisfies an IC axiom α in $\mathcal{C}$ if and only if the corresponding query $\mathcal{T}(\alpha)$ has an empty answer set:

Theorem 1 *Given an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ with expressivity $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$, we say that $\mathcal{K} \models_{IC} \alpha$ iff $\mathcal{K} \not\models \mathcal{T}(\alpha)$, where $\alpha \in \mathcal{C}$.*

Proof Sketch We enumerate all possible cases of α, which is a $\mathcal{SROIQ}$ axiom, and for each case we prove the conclusion is true. The complete proof is presented in Appendix B.□

According to Theorem 1, given an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$, the IC validation algorithm is shown as follows, which shows $\langle \mathcal{K}, \mathcal{C} \rangle$ is valid if every IC axiom α in $\mathcal{C}$ has an empty answer set:

Alg 1 IsValid($\langle\mathcal{K},\mathcal{C}\rangle$)/*check if $\langle\mathcal{K},\mathcal{C}\rangle$ is valid*/

```
for $\forall\alpha \in \mathcal{C}$ do
  $\mathcal{C} = \mathcal{C} \setminus \alpha$;
  Translate $\alpha$ to query $\mathcal{T}(\alpha)$;
  if Ans($\mathcal{T}(\alpha), \mathcal{K}$) $\neq \emptyset$ then
    return FALSE;
if $\mathcal{C} \ = \emptyset$ then
  return TRUE;
else
  return FALSE;
```

CHAPTER 6
Explanation and Repair of IC Violations

According to the IC validation approach presented in Chapter 5, given an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ with expressivity $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ ($\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$ resp.), we say that $\mathcal{K} \models_{IC} \alpha$ iff $\mathcal{K} \not\models \mathcal{T}(\alpha)$ where $\alpha \in \mathcal{C}$. That is, an IC axiom $\alpha \in \mathcal{C}$ is IC-satisfied by $\mathcal{K}$ if and only if the conjunctive query $Q(\bar{x}) = \mathcal{T}(\alpha)$ has empty answers. In other words, non-empty answers of $Q(\bar{x})$ indicate violations of the IC axiom α. Based on this, we propose an approach for explanation and repair of IC violations.

First, to explain why an IC axiom α is violated we just need to explain why the answers of conjunctive query $Q(\bar{x})$ are not empty, that is, why certain tuple $\bar{a}$ in the answer set is an answer of query $Q(\bar{x})$. Second, to repair a violation of IC axiom α is to change the KB such that the new KB will return an empty answer to query $Q(\bar{x})$.

In what follows, in Section 6.1, we first address the general problem of explanation of conjunctive query answers in OWL 2 DL (DL $\mathcal{SROIQ}$), then in Section 6.2 and 6.3, we show how to explain and repair violations of ICs based on the explanations of answers to the conjunctive queries corresponding to the given IC axioms.

6.1 Explanation of Conjunctive Query Answers in OWL 2 DL

Conjunctive query answering plays an important role in retrieving inferences from DL KBs. A significant amount of research has been devoted to study the algorithms and computational complexity of conjunctive query answering [24] [23] [38] [25]. Furthermore, in order to meet the negative inference [16] retrieval needs in realistic applications, some work on extending DL conjunctive query answering with the capability of closed-world reasoning has been conducted. The approaches include querying DL KBs with expressive query languages such as non-monotonic

[16]Negative inference refers to the failure of drawing an inference.

epistemic query languages [14] or the query languages with negation as failure [17] [64], or combining standard DL KBs with logic programming [22, 29, 40] or epistemic operators [18].

It is now commonplace to see users who do more than basic information retrieval. They require some means supporting explanation for the results they obtain. While explanation facility is provided for most standard DL reasoning tasks, it is surprisingly missing for conjunctive query answering. The core motivation for explanation of conjunctive query answers is to help users to understand the answers, especially when the answers are not expected:

- A tuple is returned as an answer but users didn't anticipate it;
- Users anticipated certain tuple to be returned as an answer but it is not;

The above two problems can be solved by explanation of positive (resp. negative) answers to conjunctive queries which explains why certain tuple of individuals is (resp. is not) an answer to the given queries.

One might think explanation of conjunctive query answer is just the process of substituting the answer variables in queries with corresponding individuals in query answers. Although this substitution process gives us some clues about the possible reasons behind query answers, we aim at finding the meaningful and precise explanations that pinpoint the root reasons for query answers. Based on this idea, explanation of positive answers to conjunctive queries could be quite complicated, especially when the queries include negative inferences retrieval at non-atomic level.

Example 16 *Given the following query*

$$Q(x) \leftarrow \mathtt{GraduateStudent}(x) \wedge \boldsymbol{not}(\mathtt{teachingAssistantOf}(x,y) \wedge \mathtt{GraduateCourse}(y) \wedge \mathtt{teacherOf}(z,y) \wedge \mathtt{Professor}(y))$$

where ***not*** *is the negation as failure for negative inference retrieval. Query $Q(x)$ retrieves all graduate students that are not teaching assistants of some graduate courses that are taught by professors.*

[17]With NAF, axioms that cannot be proven to be true are assumed to be false.

To explain why some value a is an answer to $Q(x)$ is to show why the evaluation of query body of $Q(a)$ is true. First, a has to be a graduate student. Furthermore, it is possible

- a is not a teaching assistant;
- a is a teaching assistant but the course a assists in is not a graduate course;
- a is a teaching assistant of a graduate course but the lecturer of this course is not a professor;

Besides the above ones, there are even more possible reasons: substituting variables y and z with some values, as long as the conjunction in scope of **not** , i.e., $\texttt{teachingAssistantOf}(a, y) \wedge \texttt{GraduateCourse}(y) \wedge \texttt{teacherOf}(z, y) \wedge \texttt{Professor}(y)$, is evaluated to false then we find a reason. Note that, the falseness of the above conjunction might be due to different reasons: either one of the four query atoms is false, or the conjunction of some query atoms is evaluated to be true but the combination of this conjunction with an additional query atom is false. Given a DL KB with normal expressivity and size, it would require quite a lot effort to substitute all possible values of y and z, check all possible combinations of the query atoms, and show none of the combinations is evaluated to be true. As illustrated by the above example, when the queries include negative inference retrieval, the task of explanation of positive answers to conjunctive queries requires explanations " why *none* of the possible evaluations of the query is true? " which is challenging to address if the negative inference retrieval is at non-atomic level. Similarly, the task of explanation of negative answers to conjunctive queries is also challenging since it also requires explanations related to non-atomic level of negative inference retrieval.

In this section, we present our approach on explanation of answers to conjunctive queries in DL $\mathcal{SROIQ}$. First, in Section 6.1.1 we propose the notion of justification for entailments of DCQ$^{\mathbf{not}}$ queries, and present how to explain positive answers to DCQ$^{\mathbf{not}}$ queries by reducing it to justifying corresponding query entailments; Then, in Section 6.1.2, we show explaining negative answers to DCQ$^{\mathbf{not}}$ queries can be translated to explanation of positive answers to the corresponding negated queries. In Section 6.1.3, a set of algorithms for explanation computation are provided. Finally, in Section 6.1.4 we show under certain conditions explanation of answers to normal conjunctive queries (CQ) can be reduced to the explanations of corresponding DCQ$^{\mathbf{not}}$ queries.

6.1.1 Explaining Positive Answers to DCQ$^{\mathbf{not}}$ Queries

Given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$, a DCQ$^{\mathbf{not}}$ query $Q(\bar{x})$, and query answers $\mathbf{Ans}(Q(\bar{x}), \mathcal{K}) = \{\bar{t_1}, \bar{t_2}, \ldots\}$, explanation of positive answers to $Q(\bar{x})$ is to explain why certain tuple of individuals, i.e., $\bar{t_i}$, is an answer of $Q(\bar{x})$ over $\mathcal{K}$. In other words, we need to explain why the KB $\mathcal{K}$ entails query $Q(\bar{x})$ w.r.t. some assignments that map answer variable $\bar{x}$ to tuple $\bar{t_i}$. For instance, in Example 8, query $Q(x) \leftarrow A(x) \wedge \mathbf{not}\,(p(x,y) \wedge B(y))$ has three answers (a_1), (a_2), and (a_3). To explain why the three tuples are answers of $Q(x)$ is to indicate query entailment $\mathcal{K} \models^{\sigma} Q(x)$ holds w.r.t. some assignment $\sigma : x \rightarrow a_1$ (resp. a_2, a_3).

Furthermore, it can be seen the query entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$ w.r.t. $\sigma : x \rightarrow a_i$ holds if the evaluation of query body of $Q(\bar{a_i})$ is true. With examination of $\mathcal{K}$ and $Q(x)$, we can see

- $Q(a_1)$ is true because $A(a_1)$ is true (entailed by $\mathcal{K}$), which is due to the existence of $A(a_1)$ itself in $\mathcal{K}$, and $p(a_1, y) \wedge B(y)$ is false, which is due to the truthness (entailment) of axiom $p(a_1, b)$ and falseness (non-entailment) of axiom $B(b)$;

- $Q(a_2)$ is true because $A(a_2)$ is true (entailed by $\mathcal{K}$), which is due to the existence of axioms $\{A(a_2)\}$ or $\{C \sqsubseteq A, C(a_2)\}$ in $\mathcal{K}$, and $p(a_2, y) \wedge B(y)$ is false, which is due to the non-entailment of axioms $p(a_2, y)$ and $B(y)$;

- $Q(a_3)$ is true because $A(a_3)$ is true (entailed by $\mathcal{K}$), which is due to the existence of axioms $\{\exists s.\top \sqsubseteq A, s(a_3, b)\}$ in $\mathcal{K}$, and $p(a_3, y) \wedge B(y)$ is false, which is due to the non-entailment of axioms $p(a_3, y)$ and $B(y)$.

Note that, in DLs, while axiom entailments can be explained by axiom entailment justifications, explanation of axiom non-entailments is still an open research problem due to the infiniteness: the non-entailment of an axiom could have infinite reasons. For instance, the non-entailment of axiom $B(b)$ might be due to:

- Absence of $B(b)$ from $\mathcal{K}$;
- Absence of $D(b)$ from $\mathcal{K}$ since D is a subconcept of B;

Besides the above reasons, the absence of any axioms from $\mathcal{K}$ could be a reason as long as $B(b)$ would be entailed by $\mathcal{K}$ after adding these axioms to $\mathcal{K}$. We can see most of these reasons do not make sense as explanations. By the law of Occam's Razor, to explain why an axiom is not entailed by a KB, the simplest reason, i.e., the absence of this axiom itself from the KB, should be sufficient as an explanation.

As shown by the above example, a query entailment holds because some axioms exist in the KB and some axioms are absent from the KB. These axioms are called *justifications* of conjunctive query entailments. Formally, given a query entailment $\mathcal{K} \models^{\sigma} Q$ where $\mathcal{K}$ is a DL $\mathcal{SROIQ}$ KB, Q is a DCQ$^{\mathbf{not}}$ query, σ is an assignment:

1. $\mathcal{J}_+$ is a positive justification set (positive justification, for short) for $\mathcal{K} \models^{\sigma} Q$ if $\mathcal{J}_+$ satisfies conditions a, b, c, otherwise $\mathcal{J}_+ = \emptyset$.

   (a) $\mathcal{J}_+ \subseteq \mathcal{K}$, $\mathcal{J}_+ \models^{\sigma} Q$;
   $\mathcal{J}_+$ is a fragment of $\mathcal{K}$ that entails Q w.r.t. σ.

   (b) $\forall S \subseteq \mathcal{K}$, $\mathcal{J}_+ \cup S \models^{\sigma} Q$;
   Any fragment of $\mathcal{K}$ containing $\mathcal{J}_+$ entails Q w.r.t. σ.

   (c) $\forall \mathcal{J}' \subset \mathcal{J}_+$, $\mathcal{J}' \not\models^{\sigma} Q$.
   $\mathcal{J}_+$ would not entail Q w.r.t. σ after removing any axiom from it.

2. $\mathcal{J}_-$ is a negative justification set (negative justification, for short) for $\mathcal{K} \models^{\sigma} Q$ if $\mathcal{J}_-$ satisfies conditions a, b, c, d, e, otherwise $\mathcal{J}_- = \emptyset$.

   (a) $\mathcal{K} \cap \mathcal{J}_- = \emptyset$, $\mathcal{K} \cup \mathcal{J}_-$ is consistent;
   $\mathcal{K}$ and $\mathcal{J}_-$ are disjoint, $\mathcal{K}$ is consistent after adding $\mathcal{J}_-$ to it.

   (b) $\mathcal{K} \cup \mathcal{J}_- \not\models^{\sigma} Q$;
   $\mathcal{K}$ would not entail Q w.r.t. σ after adding $\mathcal{J}_-$ to it.

   (c) $\forall T \supseteq \mathcal{J}_-$, $\mathcal{K} \cup T \not\models^{\sigma} Q$;
   $\mathcal{K}$ would not entail Q w.r.t. σ after adding axioms containing $\mathcal{J}_-$ to it.

(d) $\forall \mathcal{J}' \subset \mathcal{J}_-, \mathcal{K} \cup \mathcal{J}' \models^\sigma Q$;

$\mathcal{K}$ would still entail Q w.r.t. σ if not adding all axioms in $\mathcal{J}_-$ to it.

(e) $\mathcal{J}_-$ is $\sigma(q)$ if $Q \leftarrow \textbf{not}\, q$ where q is an atomic query atom.

$\mathcal{J}_-$ for the entailment of query **not** q w.r.t. σ is $\sigma(q)$.

3. A justification $\mathcal{J}$ for an entailment $\mathcal{K} \models^\sigma Q$ is $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle$, where $\mathcal{J}_+$ and $\mathcal{J}_-$ are positive and negative justifications respectively which are not $\emptyset$ at the same time.

According to the above definition, a justification $\mathcal{J}$ has the following properties:

- $P_{+_{suf}}$: **Sufficient Condition (existence of $\mathcal{J}_+$ in $\mathcal{K}$)** $\mathcal{J}_+$ is sufficient for the entailment to hold (by conditions 1(ab)).

- $P_{+_{min}}$: **Minimal Property (existence of $\mathcal{J}_+$ in $\mathcal{K}$)** $\mathcal{J}_+$ contains a minimal set of axioms that entail the query (by condition 1(c)).

- $P_{-_{nes}}$: **Necessary Condition (absence of $\mathcal{J}_-$ from $\mathcal{K}$)** If the entailment holds then $\mathcal{J}_-$ is absent from $\mathcal{K}$. That is, after adding $\mathcal{J}_-$ to $\mathcal{K}$ the entailment would not hold (by conditions 2(bc))

- $P_{-_{min}}$: **Minimal Property (absence of $\mathcal{J}_-$ from $\mathcal{K}$)** $\mathcal{J}_-$ contains a minimal set of axioms have to be absent from $\mathcal{K}$ (i.e., not all axioms in $\mathcal{J}_-$ can exist together) for the entailment to hold. That is, as long as not adding all axioms in $\mathcal{J}_-$ to $\mathcal{K}$ the entailment would still hold (by condition 2(d)).

The above properties show that a justification $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle$ for a DCQ$^{\textbf{not}}$ query entailment captures a minimal set of axioms existing in $\mathcal{K}$ (i.e., $\mathcal{J}_+$) and a minimal set of axioms absent from $\mathcal{K}$ (i.e., $\mathcal{J}_-$) that are sufficient and necessary conditions respectively for the given entailment to hold. Besides the above properties, it is worth mentioning that, first, $\mathcal{J}_+$ and $\mathcal{J}_-$ in a justification $\mathcal{J}$ can not be empty sets at the same time because a DCQ$^{\textbf{not}}$ query entailment holds either because the existence of a non-empty $\mathcal{J}_+$, or the absence of a non-empty $\mathcal{J}_-$, or both; second, a query entailment may have multiple positive (resp. negative) justifications, therefore multiple justifications; third, as we discussed before, we treat the absence of the axiom itself as the explanation of an axiom non-entailment, therefore, in the definition of justification, condition 2(e) specifies the justification for $\mathcal{K} \models^{\sigma} \textbf{not}\, q$ is the absence of $\sigma(q)$ form $\mathcal{K}$, where q is a query atom in form of atomic ABox axioms. That is, the justification for $\mathcal{K} \models^{\sigma} \textbf{not}\, q$ is $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle = \langle \emptyset, \{\sigma(q)\} \rangle$.

Now we use Example 8 as an instance to show how to explain query answers by justifying the corresponding query entailments. Given the KB $\mathcal{K}$ and the query $Q(x) \leftarrow A(x) \wedge \textbf{not}\, (p(x, y) \wedge B(y))$, to explain why (a_1) is an answer of $Q(x)$ over $\mathcal{K}$, first, we can see (a_1) is an answer because there is an assignment $\sigma_1 : x \rightarrow a_1, y \rightarrow b$ that maps answer variable x to the answer a_1 and the query entailment $\mathcal{K} \models Q(x)$ holds w.r.t. σ_1; furthermore, the justification for entailment $\mathcal{K} \models Q(x)$ w.r.t. $\sigma_1 : x \rightarrow a_1, y \rightarrow b$ is

$$\mathcal{J}_1 = \langle \mathcal{J}_{1_+}, \mathcal{J}_{1_-} \rangle = \langle \{A(a_1)\}, \{B(b)\} \rangle.$$

That is, the entailment $\mathcal{K} \models Q(x)$ w.r.t. σ_1 holds because $A(a_1)$ exists in $\mathcal{K}$ and $B(b)$ is missing from $\mathcal{K}$. It is easy to verify $\mathcal{J}_{1_+}$ and $\mathcal{J}_{1_-}$ satisfy all the conditions described in the justification definition. Note that, $\{A(a_1), p(a_1, b)\}$ can also entail $Q(x)$ w.r.t. σ_1 but it is not minimal therefore not a positive justification for the entailment; Also note that, with the addition of $\{B(b), B(a_1)\}$ to $\mathcal{K}$, the entailment would be false but $\{B(b), B(a_1)\}$ is not a negative justification due to the redundant axiom $B(a_1)$ in it. Similarly, (a_2) and (a_3) are answers of $Q(x)$ over $\mathcal{K}$ because the entailment $\mathcal{K} \models Q(x)$ holds w.r.t. assignment $\sigma_2 : x \rightarrow a_2$ and $\sigma_3 : x \rightarrow a_3$. The justifications for the entailment w.r.t. assignment σ_2 are

$$\mathcal{J}_{2_1} = \langle \mathcal{J}_{2_{1_+}}, \mathcal{J}_{2_-} \rangle = \langle \{A(a_2)\}, \{p(a_2, \theta), B(\theta)\} \rangle$$

$$\mathcal{J}_{2_2} = \langle \mathcal{J}_{2_{2_+}}, \mathcal{J}_{2_-} \rangle = \langle \{C \sqsubseteq A, C(a_2)\}, \{p(a_2, \theta), B(\theta)\} \rangle.$$

and the justifications for the entailment w.r.t. assignment σ_3 is

$$\mathcal{J}_3 = \langle \mathcal{J}_{3_+}, \mathcal{J}_{3_-} \rangle = \langle \{\exists s.\top \sqsubseteq A, s(a_3, b)\}, \{p(a_3, \theta'), B(\theta')\} \rangle$$

where θ and θ' are new individuals not occurring in the KB.

6.1.2 Explaining Negative Answers to DCQ$^{\mathbf{not}}$ Queries

Given a DL $\mathcal{SROIQ}$ KB $\mathcal{K}$, a DCQ$^{\mathbf{not}}$ query $Q(\bar{x})$, and query answers $\mathbf{Ans}(Q(\bar{x}), \mathcal{K}) = \{\bar{t_1}, \bar{t_2}, \ldots\}$, explanation of negative answers to $Q(\bar{x})$ is to explain why certain tuple of individuals, i.e., $\bar{t}'$, is not an answer of $Q(\bar{x})$ over $\mathcal{K}$. The requirements for explaining negative query answers usually emerge when users anticipate certain tuples to be returned as answers but they are not. This problem can be resolved by explaining positive answers to the negated queries shown as follows.

First, given a KB $\mathcal{K}$ and a query $Q(\bar{x})$, we construct the negated query of $Q(\bar{x})$, i.e., $Q'(\bar{x})$ as follows

$Q'(\bar{x}) \leftarrow \top(\bar{x}) \wedge \mathbf{not}\, Q(\bar{x})$

where $\top(\bar{x})$ is the top query atom retrieving all tuples of individuals in $\mathcal{K}$. It is easy to see $\bar{t}'$ is not an answer of $Q(\bar{x})$ if and only if $\bar{t}'$ is an answer of $Q'(\bar{x})$. To explain why $\bar{t}'$ is not an answer of $Q(\bar{x})$ we just need to explain why $\bar{t}'$ is an answer of $Q'(\bar{x})$ which can be solved by the approach of explaining positive answers to conjunctive queries presented in Section 6.1.1.

Suppose, in the KB of Example 8 there is one more individual a_4 which does not have any p successors. To explain why (a_4) is not an answer to $Q(x)$, we construct the negated query Q'

$Q'(x) \leftarrow \top(x) \wedge \mathbf{not}\,(A(x) \wedge \mathbf{not}\,(p(x, y) \wedge B(y)))$.

It is known that (a_4) is an answer of $Q'(x)$ because the query entailment $\mathcal{K} \models^{\sigma} Q'(x)$ holds w.r.t. $\sigma : x \to a_4$. That is, $\top(a_4) \wedge \mathbf{not}\,(A(a_4) \wedge \mathbf{not}\,(p(a_4, y) \wedge B(y)))$ is evaluated to be true. We can see the sub-query $\top(a_4)$ is trivially true. Furthermore, the sub-query $\mathbf{not}\,(A(a_4) \wedge \mathbf{not}\,(p(a_4, y) \wedge B(y)))$ is true either because $A(a_4) \wedge p(a_4, y) \wedge B(y)$ is true, or because $\mathbf{not}\, A(a_4)$ is true. Since a_4 does not have any p successors the latter case holds. The justification for $\mathbf{not}\, A(a_4)$ is $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle = \langle \emptyset, \{A(a_4)\} \rangle$, which is also the justification for query entailment $\mathcal{K} \models^{\sigma} Q'(x)$ w.r.t. $\sigma : x \to a_4$. Therefore, (a_4) is not an answer of $Q(x)$ because the negative justification $\mathcal{J}_-{=}A(a_4)$ is missing from $\mathcal{K}$. As long as $A(a_4)$ is added to $\mathcal{K}$ the answers of query $Q(x)$ would include an additional answer (a_4).

6.1.3 Explanation Computation: Algorithms

Based on the approach for explanation of conjunctive query answers presented in Section 6.1.1 and 6.1.2, we design a set of algorithms, i.e., Alg 1 - 6, for query answer explanation computation. The main algorithm is Alg 2. Given the inputs, i.e., a $\mathcal{SROIQ}$ KB $\mathcal{K}$, a DCQ$^{\mathbf{not}}$ query $Q(\bar{x})$, a query answer $\bar{a}$, first, the algorithm finds out the assignments σ that map $\bar{x}$ to answer $\bar{a}$ such that the entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$ holds. Then the algorithm computes the justifications for the entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$ via Alg 3, which applies assignment σ to the query body of Q, i.e., $\sigma(Q)$, then justifies the truthness of $\sigma(Q)$ via Alg 4.

Alg 4 recursively breaks down the query body of Q into the conjunction of multiple atoms A_i, computes justifications $\mathcal{J}_i$ for each atom A_i via Alg 5, then combines the justifications together via Alg 7, which combines all justifications' positive parts (PJ) together as $\mathcal{J}_+$, minimizes it to $\mathcal{J}_{min}$, then for each justification $\mathcal{J}_i$ generates a justification $\mathcal{J} = \langle\ \mathcal{J}_{min},\ \mathcal{J}_-\ \rangle$, where $\mathcal{J}_-$ is the negative part (NJ) of $\mathcal{J}_i$.

Alg 5 computes the justifications of query atom A case by case: if A is an atomic query atom q then the external algorithm JustifyAxiomEntailment is called to compute all justifications T for the entailment of axiom q, for each axiom justification $t \in T$ a justification $\mathcal{J} = \langle t, \emptyset \rangle$ is generated; if A is a **not** query atom then negative query atom justification algorithm Alg 6 is called; otherwise, A is a disjunction of multiple atoms A_i. In this case, the algorithm first computes the justifications $\mathcal{J}_i$ for each true atom A_i then combines the justifications together via Alg 7, which combines all justifications' negative parts (NJ) together as $\mathcal{J}_-$, minimizes it to $\mathcal{J}_{min}$, then for each justification $\mathcal{J}_i$ generates a justification $\mathcal{J} = \langle\ \mathcal{J}_+,\ \mathcal{J}_{min}\ \rangle$, where $\mathcal{J}_+$ is the positive part (PJ) of $\mathcal{J}_i$.

Alg 6 computes the justifications of a **not** query atom A case by case: if A is an atomic negative query atom, i.e., **not** q, additionally, if $\neg q$ is true, then the external algorithm JustifyAxiomEntailment is called to compute all justifications T for the entailment of $\neg q$ and for each justification $t \in T$ a justification $\mathcal{J} = \langle t, \emptyset \rangle$ is generated, otherwise, a justification $\mathcal{J} = \langle \emptyset, q \rangle$ is generated; if A contains duplicate **not** , i.e., **not not** A', then the justifications of A are same as A'; if A is the negation of a disjunction of multiple atoms, i.e., **not** $(A_1 \vee A_2 \vee \ldots \vee A_n)$, then according to the semantics of $\vee$ defined in Section 2.4, the justifications of A are same as justifications of **not** $A_1 \wedge$ **not** $A_2 \wedge \ldots \wedge$ **not** A_n; Otherwise, A is the negation of a conjunction of multiple atoms, i.e., **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$. The motivating example of negative inference retrieval at non-atomic level at the beginning of Section 6.1 is in this case. As discussed before, given **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$, there is a huge number of ways (2^n) to combine the n query atoms and it would require quite a lot effort to explain why none of the combinations works. With careful examination, we identify that there are three sub-cases for **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$: (1) there are **not** atoms in A_i; (2) there are no **not** atoms in A_i and all A_i atoms are evaluated to be false; (3) there are no **not** atoms in A_i and some A_i atoms are evaluated to be true, i.e., the conjunction of some atoms from A_i is evaluated to be true but with additional atoms from A_i the conjunction becomes false. It can be seen the first two sub-cases require little effort to handle and the last sub-case is the most complex case for which we found out the explanations of queries belong to this case can be optimized. Given a query **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$, instead of performing brute-force checking to all possible (2^n) combinations, here, the algorithm first checks which of the three sub-cases the query belongs to, then handles it accordingly: (1) sub-case 1: A is in form of A : **not** $(A_{p_1} \wedge \ldots \wedge A_{p_s} \wedge$ **not** $A_{n_1} \ldots \wedge$ **not** $A_{n_{n-s}})$. In this case, either $Q_a \leftarrow A_{p_1} \wedge \ldots \wedge A_{p_s} \wedge (A_{n_1} \vee \ldots \vee A_{n_{n-s}})$ or $Q_b \leftarrow$ **not** $(A_{p_1} \wedge \ldots \wedge A_{p_s})$ is true; (2) sub-case 2: all A_i atoms are positive atoms but are evaluated to be false. In this case, the algorithm computes the justifications for each **not** A_i then combines them together; (3) sub-case 3: this is the most complicated sub-case. In this case, A can be formulated as $Q_p \wedge$ **not** Q_n which is evaluated to be true, where Q_p is a conjunction of m $(m = 1, \ldots, n-1)$ atoms from A_i and Q_n is a conjunction of the remaining $n-m$ atoms from A_i. That is, some atoms from A_i form the sub-query Q_p which is evaluated to be true, but none of the answers of Q_p satisfies the subquery Q_n. It can be seen there are $\sum_{m=1,\ldots,n-1} C(n,m) = (2^n - 2)$ combinations in form of $Q_p \wedge$ **not** Q_n. To reduce the number of combinations to check without compromising the soundness of the results, we use the early termination strategy: start from checking the combinations with a sub-query Q_p of length $n-1$, continue checking the combinations with a shorter sub-query Q_p only if all combinations of current length $(n-1)$ are false. Since the explanation of many queries belonging to this sub-case will terminate in the early phase of combination checking, the performance of this algorithm is improved over the otherwise brute-fore checking approach which exhaustively checks all possible combinations.

Alg 7 combines the justifications of conjunctive or disjunctive atoms depending on the value of the input parameter cdFlag. When combining justifications of conjunctive atoms, the algorithm firsts combines all justifications' positive parts together as $\mathcal{J}_+$ and minimizes it to $\mathcal{J}_{min}$ via the helper function min, then for each justification $\mathcal{J}_i$ generates a justification $\mathcal{J} = \langle\ \mathcal{J}_{min},\ \mathcal{J}_-\ \rangle$ where $\mathcal{J}_-$ is the negative part (PJ) of $\mathcal{J}_i$. The process is similar for combining justifications of disjunctive atoms except that all justifications' negative parts are combined together as $\mathcal{J}_-$ and minimized to $\mathcal{J}_{min}$ via the function min. The helper function min($\mathcal{J}$,cdFlag,$\mathcal{K}$,Q) checks if any subsets of $\mathcal{J}$ satisfy the conditions 1(abc) (resp. 2(abcde)) in the justification definition thus are also positive (resp. negative) justifications for Q, and returns these minimal subsets.

Alg 2 ExplainQAnswer($\mathcal{K}$,$Q(\bar{x})$,$\bar{a}$)/*explain $\bar{a}$ is an answer of $Q(\bar{x})$ over $\mathcal{K}$*/

```
initialize $S$ to be an empty set of justifications;
$\bar{y}$=$Q$.vars $\setminus$ $\bar{x}$, $\sigma : \bar{x} \rightarrow \bar{a}$;
create query $Q'(\bar{y}) \leftarrow \sigma(Q.body)$, execute $Q'(\bar{y})$ over $\mathcal{K}$;
for each answer $\bar{b}$ of $Q'(\bar{y})$ do
    $\sigma$: $\{\bar{x} \rightarrow \bar{a}\} \cup \{\bar{y} \rightarrow \bar{b}\}$
    $S'$ = JustifyQEntailment($\mathcal{K}$, $Q(\bar{x})$, $\sigma$), $S \cup= S'$;
return $S$;
```

Alg 3 JustifyQEntailment($\mathcal{K}$,$Q(\bar{x})$,σ)/*justify entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$*/

```
initialize $S$ to be an empty set of justifications;
$S$ = JustifyQBody($\mathcal{K}$, $\sigma(Q)$) where $\sigma(Q)$ is the application of $\sigma$ to $Q$;
return $S$;
```

The soundness and completeness of the algorithms is shown by Theorem 2.

Theorem 2 *Let $\mathcal{K}$ be a $\mathcal{SROIQ}$ KB, $Q(\bar{x})$ be a DCQ^{not} query, $\bar{a}$ be an answer of $Q(\bar{x})$ over $\mathcal{K}$, Alg 2 - 7 together return all explanations why $\bar{a}$ is an answer of $Q(\bar{x})$ over $\mathcal{K}$.*

Alg 4 JustifyQBody($\mathcal{K}$, Q)/*justify the truthness of Q*/

```
initialize $S$ to be an empty set of justifications;
if $Q \leftarrow A$ then
  $S$ = JustifyOneConjAtom($\mathcal{K}$, $A$);
if $Q \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$ then
  initialize JList to be an empty list of sets of justifications;
  for each $A_i$ in $Q$ do
    $S_i$=JustifyQBody($\mathcal{K}$,$Q_i$) where $Q_i \leftarrow A_i$, add $S_i$ to JList;
  $S$=combineJs(JList,1, $\mathcal{K}$, $Q$);//combine justifications of conjuncts
return $S$;
```

Alg 5 JustifyOneConjAtom($\mathcal{K}$, A)/*justify $\mathcal{K} \models A$*/

```
initialize $S$ to be an empty set of justifications;
if A: q then
  $T$ = JustifyAxiomEntailment($\mathcal{K}$, q), for each $t \in T$, add $\mathcal{J} = \langle t, \emptyset \rangle$ to $S$;
if A: not A' then
  $S$ = JustifyNegAtom($\mathcal{K}$, $A$);
if $A : A_1 \vee A_2 \vee \ldots \vee A_n$ then
  initialize JList to be an empty list of sets of justifications;
  for each $A_i$ in A do
    if $Q_i(\bar{z})$ is true, where $Q_i(\bar{z}) \leftarrow A_i$, $\bar{z} = A_i$.vars then
      initialize $S_i$ to be an empty set of justifications;
      for each answer $\bar{s}$ of $Q_i(\bar{z})$ do
        $S'$=JustifyQBody($\mathcal{K}$,$\sigma(Q_i)$) where $\sigma : \bar{z} \rightarrow \bar{s}$, $S_i \cup = S'$;
      add $S_i$ to JList;
  $S$ = combineJs(JList,0, $\mathcal{K}$, $A$);//combine justifications of disjuncts
return $S$;
```

Proof Sketch To prove Theorem 2 is to prove Alg 3 - 7 together return all justifications for the query entailments $\mathcal{K} \models^{\sigma} Q(\bar{x})$. The query entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$ holds if and only if the evaluation of query body $\sigma(Q)$ is true, which is justified by Alg 4 - 7 together. Alg 4, 5, 6 together recursively break down the given query body into atomic query atoms, justify each atomic query atom, and combine them together via Alg 7. Therefore, to prove Theorem 2, we just need to enumerate the possible cases of query $Q(\bar{x})$ and for each case to prove: (1) soundness: the results returned by Alg 4 - 7 as positive (resp. negative) justifications satisfy all the conditions 1(abc) (resp. 2(abcde)) in justification definition; (2) completeness: all results satisfy the conditions 1(abc) (resp. 2(abcde)) in justification definition are returned by Alg 4

Alg 6 JustifyNegAtom($\mathcal{K}$, A)/*justify $\mathcal{K} \models A$ where A is a negative atom*/

```
initialize $S$ to be an empty set of justifications;
**if** A: **not** q **then**
  **if** $\mathcal{K} \models \neg q$ **then**
    $T$ = JustifyAxiomEntailment($\mathcal{K}$, $\neg q$), for each $t \in T$, add $\mathcal{J} = \langle t, \emptyset \rangle$ to $S$;
  **else**
    $\mathcal{J} = \langle \emptyset, \{q\} \rangle$, add $\mathcal{J}$ to $S$;
**if** A: **not not** A' **then**
  $S$ = JustifyQBody($\mathcal{K}$, $Q$) where $Q \leftarrow A'$ ;
**if** $A$ : **not** $(A_1 \vee A_2 \vee \ldots \vee A_n)$ **then**
  $S$ = JustifyQBody($\mathcal{K}$, $Q$) where $Q \leftarrow$ **not** $A_1 \wedge$ **not** $A_2 \wedge \ldots \wedge$ **not** $A_n$;
**if** $A$ : **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$ **then**
  **if** $A$ : **not** $(A_{p_1} \wedge \ldots \wedge A_{p_s} \wedge$ **not** $A_{n_1} \ldots \wedge$ **not** $A_{n_{n-s}})$/*exist **not** atoms*/ **then**
    **if** $Q_a(\bar{x})$ is true, $Q_a(\bar{x}) \leftarrow A_{p_1} \wedge \ldots \wedge A_{p_s} \wedge (A_{n_1} \vee \ldots \vee A_{n_{n-s}})$, $\bar{x}$=$A$.vars
    **then**
      **for** each answer $\bar{s}$ of $Q_a(\bar{x})$ **do**
        $S'$ = JustifyQBody($\mathcal{K}$, $\sigma(Q_a)$) where $\sigma : \bar{x} \to \bar{s}$, $S \cup = S'$;
    $Q_b(\bar{x}) \leftarrow$ **not** $(A_{p_1} \wedge \ldots \wedge A_{p_s})$, $\bar{x}$=$(A_{p_1} \ldots A_{p_s})$.vars,justify $Q_b$ same as $Q_a$;
  **else**
    **if** all $Q_i(\bar{x_i})$ are true, where $Q_i(\bar{x_i}) \leftarrow$ **not** $A_i$, $\bar{x_i}$=$A_i$.vars, i=1 ... n **then**
      initialize JList to be an empty list of sets of justifications;
      **for** each $Q_i$ (i=1 ... n) **do**
        $S_i$ = JustifyQBody($\mathcal{K}$, $Q_i$), add $S_i$ to JList;
      $S$ = combineJs(JList,0, $\mathcal{K}$, $A$);//combine justifications of disjuncts
    **else**
      **for** m= n-1 ... 1 **do**
        create all queries $Q'(\bar{x}) \leftarrow Q_p \wedge$ **not** $Q_n$ where $Q_p$ and $Q_n$ are the con-
        junction of $m$ and $n - m$ atoms from $A_i$;
        **for** each query $Q'(\bar{x})$ **do**
          **if** $Q'(\bar{x})$ is true **then**
            **for** each answer $\bar{a}$ of $Q'(\bar{x})$ **do**
              $S'$ = JustifyQBody($\mathcal{K}$, $\sigma(Q')$) where $\sigma : \bar{x} \to \bar{a}$, $S \cup = S'$;
        **if** all queries $Q'(\bar{x})$ are false **then**
          continue;
**return** $S$;
```

Alg 7 combineJs(L,cdFlag, $\mathcal{K}$, Q)/*combine justifications in $L : (S_1, \ldots, S_n)$*/

```
initialize $S$ to be an empty set of justifications;
$n = L$.size, generate all combinations $C = (\mathcal{J}_1, \ldots, \mathcal{J}_n)$ where $J_i \in S_i$;
if cdFlag == 1/*combine justifications of conjunctive atoms*/ then
  for each combination $C = (\mathcal{J}_1, \ldots, \mathcal{J}_n)$ do
    $\mathcal{J}_+ = \underbrace{\cup}_{i=1,\ldots,n} \mathcal{J}_i$.PJ;$\mathcal{J}_{min}$=min($\mathcal{J}_+$,1,$\mathcal{K}$,$Q$), add $\mathcal{J}=\langle \mathcal{J}_{min}, \mathcal{J}_i$.NJ$\rangle$ to $S$, i=1 ... n;
if cdFlag == 0/*combine justifications of disjunctive atoms*/ then
  for each combination $C = (\mathcal{J}_1, \ldots, \mathcal{J}_n)$ do
    $\mathcal{J}_- = \underbrace{\cup}_{i=1,\ldots,n} \mathcal{J}_i$.NJ;$\mathcal{J}_{min}$=min($\mathcal{J}_-$,0,$\mathcal{K}$,$Q$), add $\mathcal{J}=\langle \mathcal{J}_i$.PJ,$\mathcal{J}_{min}\rangle$ to $S$, i=1 ... n;
return $S$;
```

- 7 as positive (resp. negative) justifications. The details of the proof are presented in Appendix C.

We analyze the query complexity of the algorithms. Query complexity is the complexity measured w.r.t. the size of input query $Q(x)$. The conventional way to measure the size of a conjunctive query (CQ) is to use the number of atomic query atoms in the given CQ as the metric. For instance, given a CQ query $Q \to q_1 \wedge \ldots \wedge q_n$, where each q_i $(i = 1 \ldots n)$ is an atomic query atom, the size of Q is n. Here, the input query $Q(x)$ of the above Algorithms is a DCQ$^{\mathbf{not}}$ query which may include negative inference retrieval. We still use the number of atomic query atoms in $Q(x)$ as the size of query. For example, given a DCQ$^{\mathbf{not}}$ query $Q \to q_1 \wedge q_2 \wedge \mathbf{not}\,(q_3 \wedge q_4)$, the size of Q is 4. From the above algorithms, it can be seen, the queries of same sizes may be explained differently depending on the structures of the queries, thus require different amounts of time for explanation. Given a DCQ$^{\mathbf{not}}$ query Q of size n, we enumerate the possible cases of Q and give the query complexity of the above algorithms as follows:

- $Q \leftarrow A_1 \wedge \ldots \wedge A_n$: $O(n)$;
- $Q \leftarrow A_1 \vee \ldots \vee A_n$: $O(n)$;
- $Q \leftarrow \mathbf{not}\,(A_1 \vee \ldots \vee A_n)$: $O(n)$;
- $Q \leftarrow \mathbf{not}\,(A_1 \wedge \ldots \wedge A_n)$:

- $Q \leftarrow$ **not** $(q_1 \wedge \ldots \wedge q_s \wedge$ **not** $q_{s+1} \wedge \ldots \wedge$ **not** $q_n)$: if $q_1 \wedge \ldots \wedge q_s \wedge (q_{s+1} \vee \ldots \vee q_n)$ is true then the complexity is $O(n)$; otherwise **not** $(q_1 \wedge \ldots \wedge q_s)$ is true and the worse-case complexity is $O(2^s)$;
- All atoms A_i are false: $O(n)$;
- Otherwise, the worst case complexity is $O(2^n)$.

- $Q \leftarrow A_1 \wedge \ldots \wedge A_c \wedge$ **not** $(A_{c+1} \wedge \ldots \wedge A_n)$: the worst case complexity is $O(2^{n-c} + c)$.

where A_i is an query atom in form of q_i or **not** q_i and q_i is an atomic query atom. From the above results it can be seen that in most cases the algorithms have a linear complexity, while in the case that the query Q includes negative inference retrieval at non-atomic level the algorithms have a complexity upper bound that is exponential w.r.t. the size of the sub-query in scope of the negation. Note that, in the case of non-atomic negative inference retrieval in queries, because of the early termination strategy, the algorithms may finish early thus having a lower complexity than the upper bound.

6.1.4 Explanation of Answers to Conjunctive Queries

We have presented the explanation of answers to a category of conjunctive queries, i.e., DCQ$^{\textbf{not}}$ queries, where all query variables are distinguished and negation as failure can be used in queries for negative inference retrieval. In this section, we will show that under certain conditions, explanation of answers to general conjunctive queries (CQ) can be reduced to explanation of corresponding DCQ$^{\textbf{not}}$ queries.

It is known that a CQ query Q can be represented as a graph $\mathcal{G}(Q)$ using the rolling up approach [15] [30]: each query variable in Q is a graph node which is labeled with the corresponding concept names appearing in query atoms, and each role name in query Q is represented as a edge connecting the query variables that appear in this role query atom. If the graph $\mathcal{G}(Q)$ is tree-shaped (i.e., acyclic) then a concept C_Q corresponding to query Q can be built, and Q is entailed by a DL KB $\mathcal{K}$ if and only if $\mathcal{K} \cup \{\top \sqsubseteq \neg C_Q\}$ is inconsistent.

Example 17 *Suppose we have a conjunctive query Q*

$Q(x) \leftarrow A(x) \wedge p(x, y) \wedge B(y) \wedge q(y, z) \wedge C(z)$

where A, B, C are concept names, p, q are role names, x is a distinguished variable, y, z are non-distinguished variables. It is easy to see that query Q can be represented as a graph $\mathcal{G}(Q)$ using the rolling up approach and $\mathcal{G}(Q)$ is acyclic. Therefore a concept $A \sqcap \exists p.(B \sqcap \exists q.C)$ corresponding to Q can be built, and $\mathcal{K} \models Q$ iff $\mathcal{K} \cup \{\top \sqsubseteq \neg(A \sqcap \exists p.(B \sqcap \exists q.C))\}$ is inconsistent.

The above rolling-up approached can be used to converting a CQ query Q into a DCQ$^{\mathbf{not}}$ query Q', if in the query graph of Q, i.e., $\mathcal{G}(Q)$, the subgraph corresponding to all non-distinguished query variables and related concepts and roles is acyclic. In Example 17, $p(x, y) \wedge B(y) \wedge q(y, z) \wedge C(z)$ is a subquery of Q that contains all non-distinguished query variables and associated concepts and roles, and the subgraph in graph $\mathcal{G}(Q)$ corresponding to this subquery does not have cycles. This subquery can be expressed with concept $\exists p.(B \sqcap \exists q.C)$. Therefore the CQ query Q can be converted to a DCQ$^{\mathbf{not}}$ query $Q'(x) \leftarrow A(x) \wedge E(x)$ where concept $E = \exists p.(B \sqcap \exists q.C)$. Suppose a is an answer of $Q(x)$ over a DL KB $\mathcal{K}$, explaining why a is an answer of $Q(x)$ is actually to explain why a is an answer of query $Q'(x)$ which can be reduced to justifying corresponding query entailment $\mathcal{K} \models^{\sigma} Q'(x)$ w.r.t. σ: $\{x \rightarrow a\}$, which returns justification $\mathcal{J} = \langle \{A(a), E(a)\}, \emptyset \rangle$. That is, a is an answer of $Q(x)$ because of the existence of axioms $\{A(a), E(a)\}$ (i.e., $\{A(a), (\exists p.(B \sqcap \exists q.C))(a)\}$) in $\mathcal{K}$. Converting a conventional CQ including non-distinguished variables to a distinguished conjunctive query only including distinguished variables is a broad topic and could be very complicated. Here, we just use the above example to show that under certain conditions conventional CQ queries can be translated to DCQ$^{\mathbf{not}}$ queries and the explanations of answers to conventional CQ queries can be reduced to explaining answers to the corresponding DCQ$^{\mathbf{not}}$ queries.

6.2 Explanation of IC Violations

Based on the IC semantics of OWL 2 DL and the IC validation approach presented in Chapter 4 and 5, given an extended KB pair $\langle \mathcal{K}, \mathcal{C} \rangle$, an IC axiom α in

$\mathcal{C}$ is IC-satisfied by $\mathcal{K}$ if and only if the conjunctive queries corresponding to axiom α have empty answers.

Example 18 *Suppose we have the following KB $\mathcal{K}$ and an integrity constraint axiom α*

$$\mathcal{K} = \{\texttt{Zinfandel} \sqsubseteq \texttt{Wine}, \texttt{Zinfandel}(p_1), \texttt{Wine}(p_1), \texttt{locatedIn}(p_1, s_1), \texttt{locatedIn}(p_1, s_2)\}$$

$$\alpha : \texttt{Wine} \sqsubseteq \exists \texttt{locatedIn}.\texttt{Region}.$$

Where the IC axiom α is translated into conjunctive query Q

$$Q(x) \leftarrow \texttt{Wine}(x) \wedge \textit{\textbf{not}}\,(\texttt{locatedIn}(x, y) \wedge \texttt{Region}(y))$$

Since query $Q(x)$ has an answer p_1 an IC violation is detected.

To explain the above IC violation, we just need to explain why p_1 is an answer of $Q(x)$ over $\mathcal{K}$, which can be reduced to the computation of justifications for query entailment $\mathcal{K} \models Q(x)$ w.r.t. the following assignments:

$$\sigma : x \rightarrow p_1, y \rightarrow s_1$$

$$\sigma' : x \rightarrow p_1, y \rightarrow s_2$$

$$\sigma'' : x \rightarrow p_1, y \rightarrow p_1$$

For example, the justifications for $\mathcal{K} \models Q$ w.r.t. σ are:

$$\mathcal{J}_+{}^{All} = \{\mathcal{J}_{1+}, \mathcal{J}_{2+}\} = \{\{\texttt{Wine}(p_1)\}, \{\texttt{Zinfandel}(p_1), \texttt{Zinfandel} \sqsubseteq \texttt{Wine}\}\},$$

$$\mathcal{J}_-{}^{All} = \{\mathcal{J}_-\} = \{\{\texttt{Region}(s_1)\}\},$$

$$\mathcal{J}_1 = \langle \mathcal{J}_{1+}, \mathcal{J}_- \rangle = \langle \{\texttt{Wine}(p_1)\}, \{\texttt{Region}(s_1)\} \rangle,$$

$$\mathcal{J}_2 = \langle \mathcal{J}_{2+}, \mathcal{J}_- \rangle = \langle \{\texttt{Zinfandel}(p_1), \texttt{Zinfandel} \sqsubseteq \texttt{Wine}\}, \{\texttt{Region}(s_1)\} \rangle,$$

where $\mathcal{J}_+{}^{All}$ and $\mathcal{J}_-{}^{All}$ are the collections of positive and negative justifications respectively, and $\mathcal{J}_1$ and $\mathcal{J}_2$ are two justifications.

The entailment $\mathcal{K} \models^{\sigma} Q$ contributes to the non-empty answers of query Q, since non-empty query answers suggest IC violations, the violation of the IC α can be explained by the justifications for the entailment $\mathcal{K} \models^{\sigma} Q$. Therefore, we can say, $\mathcal{K}$ violates the IC α because p_1 is a wine (due to $\mathcal{J}_{1+}$ or $\mathcal{J}_{2+}$) and p_1 does not have a known location (due to $\mathcal{J}_-$). Similarly we can get the justifications for $\mathcal{K} \models Q$ w.r.t. assignments σ' and σ'' which also explain why the IC α is violated by $\mathcal{K}$.

6.3 Repair of IC Violations

An IC is violated because the conjunctive query corresponding to this IC has non-empty answers. That is, there exist assignments such that the corresponding query entailments holds. Therefore we can assume that if the corresponding query entailments do not hold any more, then the IC violation will be repaired. Given a query entailment, the problem of *query entailment invalidation* is to identify how to change the KB such that the query entailment does not hold any more.

Before proceeding to solutions to query entailment invalidation, we first introduce the notion of hitting sets. Given a collection of sets, a set which intersects all sets in the collection in at least one element is called a hitting set (HS). A minimal hitting set (MHS) is a hitting set of minimal size. For example, suppose we have a collection of three sets $C = \{S_1 = \{a, b, c\}, S_2 = \{b, e, f\}, S_3 = \{c, g\}\}$, then there are four MHSs of C, which are MHS$_1$ = $\{b, c\}$, MHS$_2$ = $\{b, g\}$, MHS$_3$ = $\{c, e\}$, MHS$_4$ = $\{c, f\}$.

Now, the solution to query entailment invalidation can be obtained by the following observation

Given a query entailment $\mathcal{K} \models^{\sigma} Q$ where $\mathcal{K}$ is a $\mathcal{SROIQ}$ KB, Q is a DCQ$^{\mathbf{not}}$ query, σ is an assignment, suppose $\mathcal{J_+}^{All}$ and $\mathcal{J_-}^{All}$ are the collections of all positive and negative justifications respectively for the given entailment. To invalidate the entailment $\mathcal{K} \models^{\sigma} Q$, we can update the KB $\mathcal{K}$ to a new KB $\mathcal{K}$' as follows:

- $\mathcal{K}' = \mathcal{K} \setminus m$ where m is a MHS of $\mathcal{J_+}^{All}$;
- Or $\mathcal{K}' = \mathcal{K} \cup \mathcal{J}_-$ where $\mathcal{J}_- \in \mathcal{J_-}^{All}$.

The above observation is a straightforward conclusion from the properties $P_{+_{suf}}$ and $P_{-_{nes}}$ of query entailment justifications described in Section 6.1.1. Because of property $P_{+_{suf}}$, a positive justification is sufficient for a query entailment to hold. After removing a MHS of $\mathcal{J_+}^{All}$ from $\mathcal{K}$, none of the positive justifications would exist, therefore the given query entailment would become false; Because of property $P_{-_{nes}}$, after adding a negative justification $\mathcal{J}_-$ to $\mathcal{K}$, the given query entailment would not hold. Note that, a query entailment $\mathcal{K} \models^{\sigma} Q$ can also be invalidated by removing all positive justification axioms, i.e., $\mathcal{J_+}^{All}$, from $\mathcal{K}$. However this will remove more axioms than necessary and it may cause unwanted changes to the KB. Intuitively, minimal impacts on the KB are expected. This MHS-based solution is even more reasonable when the size of ${\mathcal{J}_+}^{All}$ is large.

Consider Example 18, there are two MHSs of ${\mathcal{J}_+}^{All}$, which are H_1 and H_2

$H_1 = \{\texttt{Wine}(p_1), \texttt{Zinfandel} \sqsubseteq \texttt{Wine}\}$, $H_2 = \{\texttt{Wine}(p_1), \texttt{Zinfandel}(p_1)\}$.

To invalidate the entailment $\mathcal{K} \models^\sigma Q(x)$, we can update $\mathcal{K}$ to one of the following KBs:

$\mathcal{K}'_1 = \mathcal{K} \setminus H_1 = \{\texttt{Zinfandel}(p_1), \texttt{locatedIn}(p_1, s_1),$
$\texttt{locatedIn}(p_1, s_2)\}$,

$\mathcal{K}'_2 = \mathcal{K} \setminus H_2 = \{\texttt{Zinfandel} \sqsubseteq \texttt{Wine}, \texttt{locatedIn}(p_1, s_1),$
$\texttt{locatedIn}(p_1, s_2)\}$,

$\mathcal{K}'_3 = \mathcal{K} \cup \mathcal{J}_- = \{\texttt{Zinfandel} \sqsubseteq \texttt{Wine}, \texttt{Zinfandel}(p_1), \texttt{Wine}(p_1), \texttt{locatedIn}(p_1, s_1),$
$\texttt{locatedIn}(p_1, s_2), \texttt{Region}(s_1)\}$.

It can be seen that $\mathcal{K}'_i \not\models^\sigma Q(x)$ (i=1, 2, 3). Similarly we can invalidate $\mathcal{K} \models Q$ w.r.t. assignments σ' and σ''. It is easy to verify that the answers to $Q(x)$ over new KBs are empty and there are no IC violations any more.

CHAPTER 7
Implementation and Evaluation

In this chapter, we present the implementation and evaluation details of the OWL 2 DL-based approach that we have proposed for ICs on the Semantic Web. In Section 7.1 we introduce a prototype implementation which includes the services for IC validation, IC violation explanation and repair that we have described in Chapter 5 and Chapter 6. Then, in Section 7.2 we describe the evaluation approach that we have performed on some Semantic Web instance data, and the results that we have found. A performance analysis of our system is presented in Section 7.3.

7.1 Implementation

We have developed a prototype implementation for the OWL 2 DL-based approach that we have proposed for the ICs on the Semantic Web. The system reads the input instance data and OWL IC axioms, reports the IC violations detected, explains the violations, computes the repairs and applies the repairs to fix the violations. Given the inputs, the system firsts materializes the inferences on the instance data and saves the inference result as a rule base using the *inference materializer*, and converts the OWL IC axioms into a set of rules using the *IC converter*; then it executes the rules over the rule base to detect IC violations using the *IC validator*; for each violation, it generates the explanations using the *IC violation explainer*, computes and applies the repairs using the *IC violation repairer*. The details of the implementation are as follows:

- **Inference Materializer:** the inference materializer accepts the input instance data as an OWL KB, computes the DL inferences over the KB, extracts the ABox assertions and saves them as the facts in a rule base.

- **IC Converter:** the converter loads the input OWL IC axioms, converts them into a set of rules. Each IC is represented as a rule where the rule body is obtained according to the translation rules described in Section 5.1 and the rule head is a literal with a special predicate. If the rule body is evaluated to be true then the rule head is true which suggests the IC is violated.

- **IC Validator:** the validator accepts the input rule base (i.e., the rule representation of the input instance data) and the rules (i.e., the rule representation of the input OWL ICs), treats them as a whole logic program, computes the model(s) of the logic program, retrieves the individuals from the literals in the model(s) and returns them as the violating individuals (i.e., individuals that violate the ICs). We use a Java-based implementation [18] of the DLV system [19] as the rule engine. Note that, this rule engine-based implementation for IC validation is equivalent to the conjunctive query answering-based IC validation approach that we have described in Chapter 5. It is easy to verify that an OWL DL KB $\mathcal{K}$ IC-satisfies the IC axiom α), i.e., $\mathcal{K}$ returns empty answers to the conjunctive query $\mathcal{T}(\alpha)$, iff the logic program P, that consists of the rule base obtained from the DL inference materialization of $\mathcal{K}$ and the rules converted from α, has an empty model. That is, P has non-empty models iff α is violated by $\mathcal{K}$. Therefore, the individuals retrieved from the literals in the models of P are exactly the individuals that violate α.

- **IC Violation Explainer:** the explainer computes all explanations to a given IC violation. It is based on an general implementation of conjunctive query answer explainer that explains why a given tuple is an answer of the conjunctive query over an OWL DL KB. It utilizes the axiom entailment explanation functionality in OWL reasoners to get the explanations to the grounded atomic query atoms and combines them together as explanations to the whole conjunctive query according to the algorithms described in Section 6.1.3.

- **IC Violation Repairer:** given the explanations generated by the IC violation explainer, the repairer computes all possible repairs to the violation. It is based on an general implementation of conjunctive query answer invalidator which computes all invalidations of a query answer such that the the given

[18]DLV Wrapper: http://www.dlvsystem.com/dlvsystem/index.php/DLV_WRAPPER
[19]DLV System: http://www.dlvsystem.com/dlvsystem/index.php/Products

conjunctive query no longer has the answer after applying the invalidations to the KB (as shown in Section 6.3). A repair is either an addition of some axioms to the KB or a deletion of some axioms from the KB. Given the input user preferences, different repairs can be applied to the input KB to fix the violations.

7.2 Evaluation of Instance Data

To demonstrate the effectiveness of our approach in discovering the defects in Semantic Web instance data, we have evaluated some instance data from the real world using our approach. For each instance data, we selected some ICs and validated the ICs to see if they are violated by the instance data.

Datasets: we have chosen five different datasets, as shown in Table 7.1, as the input instance data to be evaluated.

Instance Data	**Pointer**
Wine	http://wineagent.tw.rpi.edu/restaurants/
MLSO	http://escience.rpi.edu/2010/mlso/PML/
CEDAR **CEDAR Time**	http://escience.rpi.edu/ontology/vsto/2/0/cedar.owl http://escience.rpi.edu/ontology/cedar/cedar-time.tgz
BCODMO	http://escience.rpi.edu/ontology/BCO-DMO/1/0/
Datagov	http://logd.tw.rpi.edu/vocab/Dataset http://purl.org/twc/vocab/conversion/MetaDataset

Table 7.1: Instance data to evaluate

Instance Data	**ICs from Ontologies**	**Additional ICs**
Wine	8 [167]	N/A
MLSO	35 [58]	7
CEDAR **CEDAR Time**	30 [47] 3 [47]	N/A 12
BCODMO	7 [47]	2
Datagov	N/A	4

Table 7.2: ICs for all instance data

ICs: as we have discussed in Section 1.1.2.3 and Section 1.1.3, a lot of users of the Semantic Web data expect that various axioms such as RDFS axioms or OWL restriction axioms in ontologies can be used as ICs. In this evaluation, we focus on reusing the axioms in the referenced ontologies as the input ICs. For each of the instance data shown in Table 7.1, we identified the axioms in its referenced ontologies (if the ontologies exists) that people require or highly expect them to be used as ICs. Our approach supports any legal RDFS axioms and OWL 2 DL axioms to be used as ICs. When identifying IC axioms from the referenced ontologies, only axioms including various OWL restrictions are considered. This is because, most axioms that people want to use as ICs are OWL restrictions, while usually RDFS domain, range axioms, and non-OWL-restriction axioms are used as inference axioms. Besides the ICs from ontologies, for some datasets, we also collected additional ICs that are required by users. This process of IC collection is aided by different people such as ontology authors, instance data authors, application developers, and instance data users. The summary of ICs for each instance data is shown in Table 7.2.

- ICs for Wine data: the wine data is used by the wine agent [20] which recommends suitable wines for different foods. The recommendation is based on different properties of wines such as color, sugar, body, and flavor. Therefore it is important to make sure the wine instances have some values specified for these properties. Among the 167 OWL restrictions in the wine ontology [21], there are 8 restrictions, including 4 `owl:allValuesFrom` and 4 `owl:cardinality` on these properties which are treated as ICs. The details of ICs are shown in Table 7.3.

- ICs for MLSO data: the main referenced ontologies of MLSO data are PML ontologies [22], i.e., ontologies for the representation of the provenance and the justification information. In the MLSO data, it is important to make sure the representation of various information is in an acceptable state. For example, a `NodeSet` should have exactly one conclusion, an `InferenceStep` should have at most one inference rule, one inference engine, and one source usage, a

[20]Wine agent: http://wineagent.tw.rpi.edu/

[21]Wine ontology (wine agent version): http://wineagent.tw.rpi.edu/data/wine.rdf

[22]PML ontologies:
http://inference-web.org/2.0/pml-provenance.owl
http://inference-web.org/2.0/pml-justification.owl

`InferenceRule` should have at most one content, a `SourceUsage` should have exactly one source, etc. These restrictions in the ontology are highly desired to be treated as ICs. We have identified 35 ICs from the 58 restrictions in the PML ontologies. Besides the ICs from PML ontologies, we have also indentified 7 ontology-independent ICs that are required for PML instance data validation. The details of ICs are shown in Table 7.4.

- ICs for CEDAR and CEDAR Time data: the main referenced ontology is the VSTO ontology [23] which is an ontology of virtual solar-terrestrial observatory. VSTO ontology provides a set of vocabulary for the description of data products, datasets, deployments, instruments, observatories, parameters, etc., and various relationships among them. Given a VSTO instance data, it is important to make sure that the data contains required information. For example, different `DataProduct` such as `HeightVsTimePlot`, `TimeSeriesPlot` and `GeoPlot` should have corresponding plotted parameters; a `Dataset` should be associated with one instrument and one instrument operating mode, and it should have some contained parameters; a `Deployment` should have one ID; a `GroundBasedInstrument` should have one longitude and one latitude; different `Parameter` such as `TimeDependentParameter`, `AuroralBoundaryIndex`, `LongitudeDependentParameter`, `LatitudeDependentParameter`, and `Altitude DependentParameter` should have corresponding coordinates; the super physical domains of `PhysicalDomain` such as `AtmosphereLayer` and `SurfaceLine` should have required types, etc. From the 47 restrictions in the VSTO ontology, We have identified 30 ICs for CEDAR data validation and 3 ICs for CEDAR Time data validation. Besides the ICs from VSTO ontology, we also identified 12 additional ICs for CEDAR Time data validation as follows. The details of ICs are shown in Table 7.5 and Table 7.6.

- ICs for BCODMO data: BCODMO data is generated using the BCODMO ontology [24] which is an ontology of biological and chemical oceanography data management. For better use of the instance data it should be in a legal state.

[23]VSTO ontology: http://escience.rpi.edu/ontology/vsto/2/0/vsto.owl
[24]BCODMO ontology: http://escience.rpi.edu/ontology/BCO-DMO/1/0/bcodmo_classes.owl

For instance, a `Deployment` should have exactly one id; a `Dataset` should have one name; a `DeploymentDataset` should be from at most one deployment; the `Inventory` of `DeploymentDatasetCollection` should be `DeploymentDataset`; an `Award` should have one award number; a `Person` should have one first name and one last name, etc. Among the 47 restrictions in BCODMO ontology, we have identified these 7 ICs. Besides the ICs from BCODMO ontology, there are 2 additional ICs. The details of ICs are shown in Table 7.7.

- ICs for Datagov data: this evaluation is only for the purpose of S2S dataset listing work which requires various datagov datasets to have certain property values so that the datasets can be correctly displayed. For example, an `AbstractDataset` should have a `title`, a `description`, a `source_agency`, and a `dataset_category`. These 4 ontology-independent ICs are represented. The details of ICs are shown in Table 7.8.

ICs from onto.	$\texttt{Wine} \sqsubseteq = 1\texttt{hasBody}.\top$, $\texttt{Wine} \sqsubseteq = 1\texttt{hasColor}.\top$, $\texttt{Wine} \sqsubseteq = 1\texttt{hasFlavor}.\top$, $\texttt{Wine} \sqsubseteq = 1\texttt{hasSugar}.\top$, $\texttt{Wine} \sqsubseteq \forall\texttt{hasBody}.\texttt{WineBody}$, $\texttt{Wine} \sqsubseteq \forall\texttt{hasColor}.\texttt{WineColor}$, $\texttt{Wine} \sqsubseteq \forall\texttt{hasSugar}.\texttt{WineSugar}$, $\texttt{Wine} \sqsubseteq \forall\texttt{hasFlavor}.\texttt{WineFlavor}$
Additional ICs	N/A

Table 7.3: ICs for wine data

Results: given the above input instance data and ICs, we have validated the instance data against the ICs. The result is shown in Table 7.9.

- Violations in wine data: among the 8 ICs, 6 ICs are violated by different individuals; totally, there are 51 violations; for example, some individuals used as values for property `hasColor` are not typed as `WineColor`, such as `Golden`, `Pink`, `Purple`, `Yellow`; misusage of the `WineBody` instance `Medium` and the `WineFlavor` instance `Moderate`; the individual `Weak` used as the value for property `WineFlavor` is not typed as `WineFlavor`; some wine instances miss values for the 4 properties, i.e., color, sugar, body, and flavor.

- Violations in MLSO data: all ICs are satisfied by the MLSO data except one ontology-independent IC which is violated by 53 individuals. This ontology-independent IC requires that each `Information` instance, which is a conclusion of some `NodeSet`, to have some values for `hasFormat` property. In the given MLSO data, all such `Information` instances miss `hasFormat` assertions.

- Violations in CEDAR and CEDAR Time data: (1) among the 30 ICs, 6 ICs are violated by the Cedar data; totally, there are 111 violations; for example, some `DataProduct` such as `HeightVsTimePlot` and `TimeSeriesPlot` miss plotted parameters; some `Parameter` such as `TimeDependentParameter`, `AuroralBoundaryIndex`, `AltitudeDependentParameter` miss time or altitude coordinates; the super physical domains of some `SurfaceLine` are not `Surface`. (2) among the 15 ICs, 3 ICs are violated by the CEDAR Time data; totally, there are 804 violations; for instance, some `Dataset` have more than one value for property `isFromInstrument`; some `Dataset` have more than one value for property `isFromInstrumentOperatingMode`; some `Dataset` miss values for property `hasContainedParameter`.

- Violations in BCODMO data: among the 9 ICs, there are 2 ICs violated by the BCODMO data; totally, there are 2546 violations; some `DeploymentDataset` miss time coverage or parameters.

- Violations in Datagov data: all 4 ICs are violated by the same batch of `AbstractDataset` which miss values for `title`, `description`, `source_agency`, `dataset_category`; totally, there are 892 violations.

It can be seen that a considerable ICs have been violated by a large number of individuals in the instance data. Leaving these violations in the data without repairing them and notifying the users may result in application failures. In the scenarios that data correctness and integrity is important, these violations may cause severe problems.

Discussions: Besides the above results, we also have the following findings:

- Restrictions vs. ICs: Using an OWL restriction as an IC is different from using it as a normal restriction: an IC axiom enforces the legal or acceptable state of the data while an restriction axiom generates inferences on the data. For example, in the wine ontology, treating the `owl:allValuesFrom` restriction on the property `hasColor` as an IC will enforce all individuals that are used as values for property `hasColor` are `WineColor`, while using it as a normal restriction will infer that any individuals used as values for property `hasColor` are `WineColor`. In this instance data evaluation, we have identified a large number of ICs from the restrictions in the ontologies ($\geq 50\%$ for PML, VSTO) that people expect them to be used as ICs. It shows we need to be very careful to distinguish whether an axiom should be used as an inference axiom for reasoning or as an IC axiom for constraint validation. It may vary in different scenarios, applications, and users.

- Ontology evaluation vs. Instance data evaluation: in the process of collecting ICs from the axioms in the ontologies, we examine every restriction and people suggest that some restrictions in the ontologies should be updated/removed. 1) in PML ontology, there are 6 such suggestions: for instance, a `NodeSet` can have more than one values for `isExplanationOf`, an `IdentifiedThing` can have multiple names and owners; an `InferenceEngine` does not need to have some inference engine rules, etc. 2) in VSTO ontology, there are 6 such suggestions: for example, the individual `EarthSurface` is mistyped and it should be typed as `Surface` instead of `SurfaceRegion`; a `Dataset` can connect to `Instrument`, `Index`, `Model` and it can only connect to one of the three entities at one time; `Atmosphere` have at least one of the three states `ElectronState`, `NeutralState`, `IonState`; there is no connection between `Wavelength` and time coordinates, etc. 3) in BCODMO ontology, there are 2 suggestions: a `Deployment` can have more than one platforms; a `DeploymentDataset` can be from multiple instruments. These findings show that the process of instance data evaluation may help the evaluation of ontologies which need to evolve to adapt to the changes in domain knowledge, applications or datasets.

- Mapping ICs in databases to OWL IC axioms: when the data is migrated from the databases to the Semantic Web, it is quite often to see the requirements to represent or map the ICs from databases using some knowledge representations on the Semantic Web. For example, the BCODMO ontology is generated after the BCODMO database schema which contains more than 100 ICs. The ontology and the database schema are supposed to represent the BCODMO domain knowledge consistently. However, in the process of evaluation, we have found that: 1) only a small portion of ICs in BCODMO databases are modeled as restrictions in BCODMO ontology; 2) even the same category of ICs in the databases are modeled as different OWL restrictions; 3) some OWL restriction representations of database ICs are not appropriate. To represent and map the database ICs to OWL IC axioms, we need a correct and consistent approach. A proposal for this mapping is shown in Table 7.10 which is based on MySQL databases. Note that, the uniqueness can be straightforwardly expressed using a single rule.

7.3 Performance Analysis

For the performance analysis, we used the datasets of different size and modeled ICs for the data. First, the validation service is performed to discover the violations in the datasets. Then, for each violation detected, the violation justification service is executed to compute justifications (and minimal justifications). In this section, we evaluate the IC validation time, i.e., the time it takes to validate a single IC over a dataset, the IC justification time, i.e., the time it takes to compute all justifications (and all minimal justifications) for a single violation. To show the effects of referenced ontologies on the performance, the referenced ontology loading is either turned on or turned off.

Datasets: we have chosen 10 LUBM [25] datasets that represent 1 to 10 universities. The datasets are generated using the LUBM data generator that produces synthetic OWL instance data over the University ontology [26]. The size of the datasets scales from around 100K to more than 1 million triples. The details of the LUBM datasets are shown in Table 7.11.

[25]LUBM benchmark: http://swat.cse.lehigh.edu/projects/lubm/

[26]University ontology: http://www.lehigh.edu/ zhp2/2004/0401/univ-bench.owl

ICs: we modeled five representative ICs shown in Table 7.12. Note that, first, these ICs are only for the purpose of performance evaluation. It is possible that some of the ICs are not suitable for real instance data evaluation. Second, to show the influence of different ICs on the performance, these ICs vary in the size and the complexity but they are not aimed to represent all possible ICs. A comprehensive evaluation would need more representative ICs to be modeled.

Results: we evaluate the IC validation time, the IC violation justification time, of the ICs over datasets of different sizes, where the referenced ontology loading is switched on or off.

- Validation Time: the validation time of the ICs over 10 datasets when the referenced ontology loading is turned on/off is shown in Figure 7.1. It can be seen that: 1) the validation time linearly increases as the dataset size increases; when the referenced ontologies are not loaded, it takes about 1 second in the dataset of 1 university and increases to 13 seconds in the dataset of 10 universities; 2) it takes longer time for validation when the referenced ontologies loading is switched from off to on. For instance, in the dataset of 1 university, the validation time increases from 1 second to 1.5 second. This is due to the query answering-based nature of validation. That is, validation time is essentially the time to execute the queries corresponding to the ICs. When the referenced ontologies are loaded, the rule base obtained from the inference materialization of the input instance data becomes larger. Therefore, the query answering time over the rule bases increases. 3) it takes almost the same time for different ICs to be validated. Since the query execution time is almost same for the different ICs they have similar duration of time for validation. In summary, the validation time is affected by the dataset size and the reasoning setting.
- Justification Time: 1) the justification time of the IC violations over 10 datasets when the referenced ontology loading is turned off is shown in Figure 7.2. We can see, that similar to the validation time, the justification time also linearly increases as the dataset size increases; among the 5 ICs, IC_4 takes the longest time to compute justifications and the other ICs take almost the

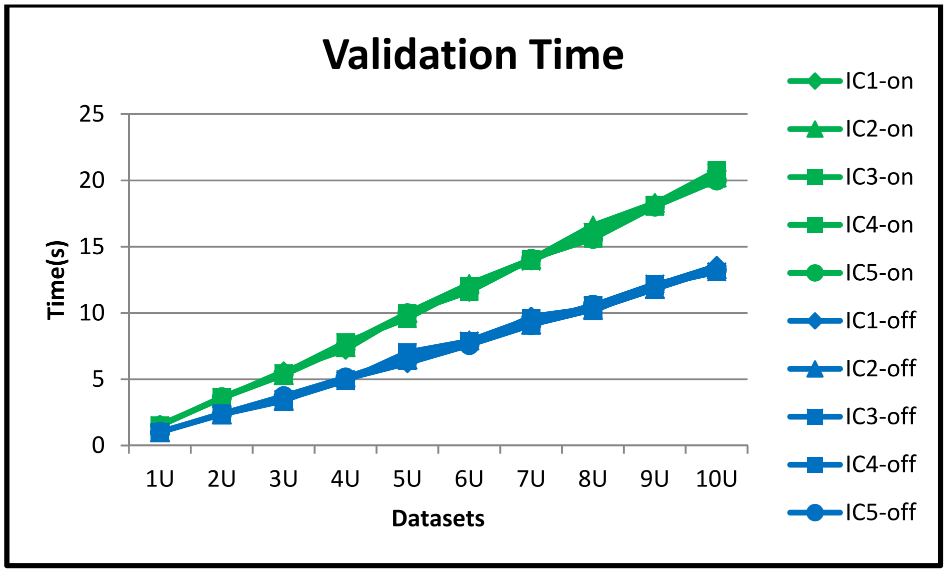

Figure 7.1: Validation time

same amount of time. This is because, IC violation justification calls axiom entailment justification, and the conjunctive query corresponding to IC_4 requires more calls to the axiom entailment justification function while the other ICs requires less. 2) the justification time for IC violations in the dataset of 1 university, when the referenced ontology loading is switched from off to on, is shown in Figure 7.3. It can be seen that justification time for violations of all ICs increases when the referenced ontology loading is turned on. This is because, when the referenced ontology loading is switched from off to on, the axiom entailment justification function would take longer duration of time to compute the justifications: when the referenced ontology loading is turned off, the justification only involves the axioms in the instance data; while the referenced ontology loading is turned on, the justification involves not only the axioms in the instance data but also the axioms in the referenced ontologies, which may result in more justifications to be returned for the same axiom entailment. Additionally, when the referenced ontology loading is switched from off to on, while the justification time for all ICs increases, IC_1 and IC_2 increases the most and they take the longest time. This is because, when the referenced ontology loading is turned on, class `Publication` used in IC_1 and IC_2 have more related axioms, such as the domain and range axioms in the referenced ontologies. These related axioms, when they are combined with the related property assertions in the instance data, would lead to more possibilities for the entailment of the membership of an individual in class `Publication`, thus more justifications for violations of IC_1 and IC_2. 3) we distinguish justifications from minimal justifications which do not contain redundant axioms. The justification time and minimal justification time for IC violations in the dataset of 1 university, is shown in Figure 7.4. It can be seen that the time to compute minimal justifications for violations of IC_5 is much longer than the time to compute its justifications, while it takes almost the same time to compute the justifications and minimal justifications for other ICs. This is due to the non-minimality of IC_5 justifications which requires extra processing, such as DL reasoning and some post processing, to remove the redundant axioms. The complete results about the justification time and the minimal justification time of all IC violations over 10 datasets when the referenced ontology loading is turned off and on is shown in Figure 7.5 and Figure 7.6 respectively. In summary, the justification time is affected not only by the dataset size and reasoning setting, but also by the ICs, since different ICs many require different times of calls to the axiom entailment justification functione.

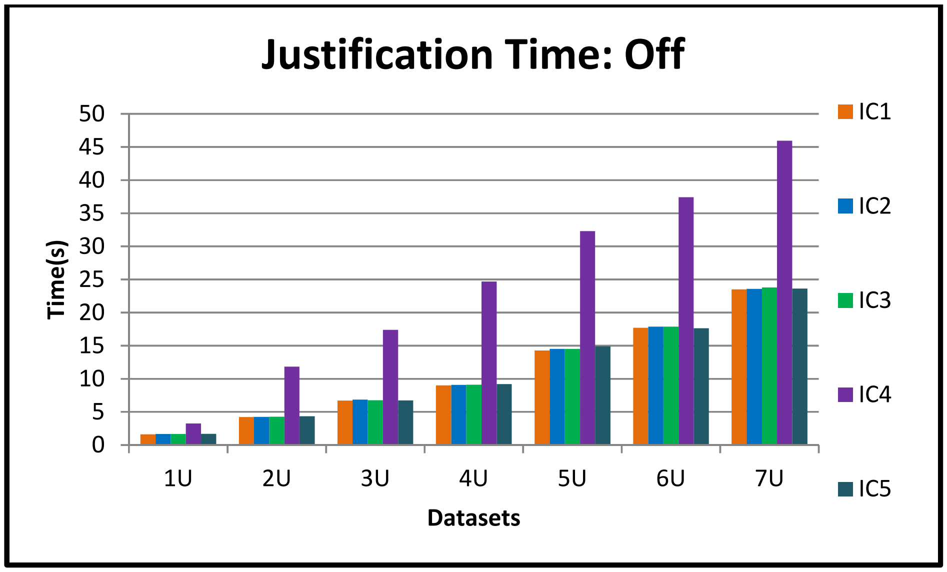

Figure 7.2: Justification time: off

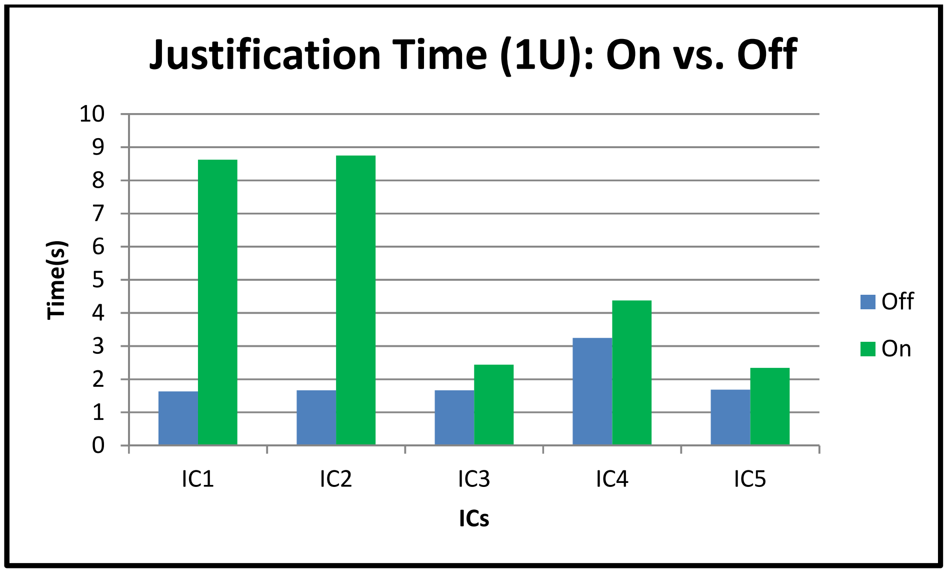

Figure 7.3: Justification time (1U): on vs. off

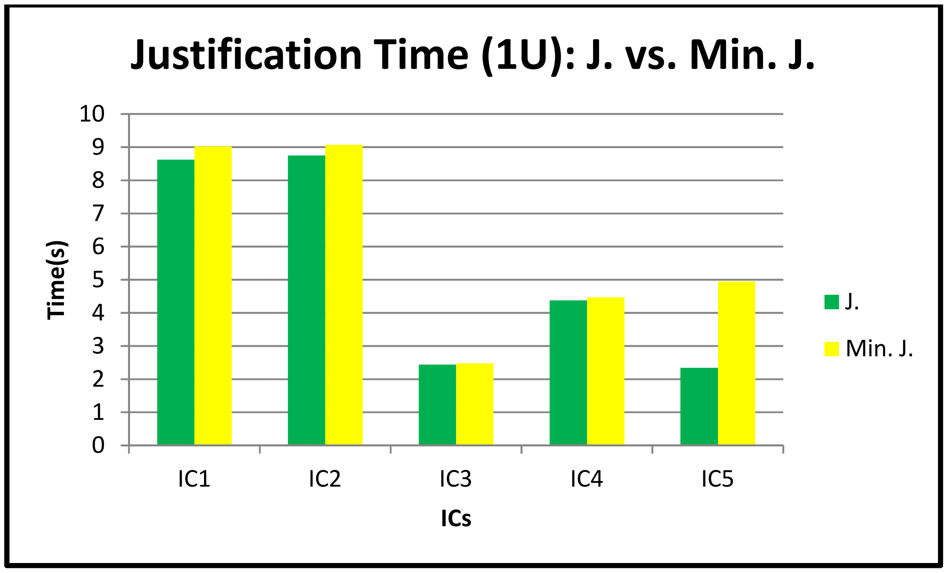

Figure 7.4: Justification time (1U): justification vs. minimal justification

The above performance evaluation results about the validation time and the justification time show that our approach can be used to validate ICs and generate explanations for IC violations in reasonable amount of time.

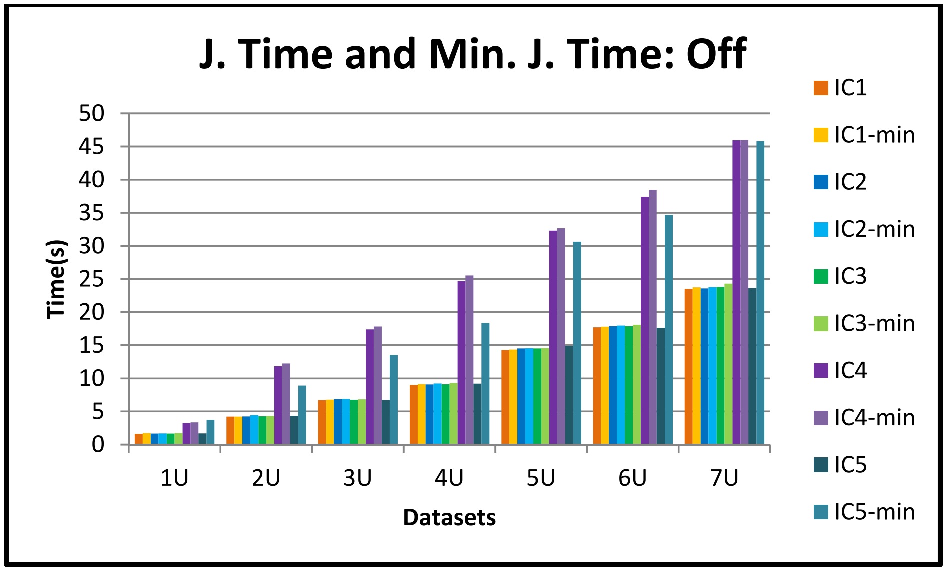

Figure 7.5: Justification time and minimal justification time: off

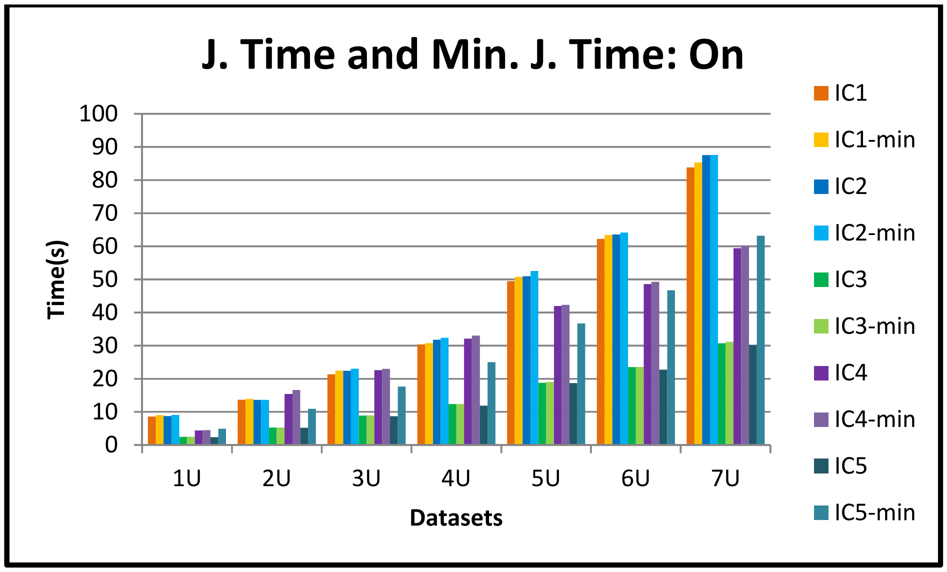

Figure 7.6: Justification time and minimal justification time: on

ICs from onto.	$\texttt{Document} \sqsubseteq \leq 1\texttt{hasContent}$ $\texttt{Document} \sqsubseteq \leq 1\texttt{hasVersion}$ $\texttt{Document} \sqsubseteq \leq 1\texttt{hasAbstract}$ $\texttt{DocumentFragment} \sqsubseteq = 1\texttt{hasDocument}$ $\texttt{DocumentFragmentByOffset} \sqsubseteq = 1\texttt{hasFromOffset}$ $\texttt{DocumentFragmentByOffset} \sqsubseteq = 1\texttt{hasToOffset}$ $\texttt{DocumentFragmentByRowCol} \sqsubseteq = 1\texttt{hasToRow}$ $\texttt{DocumentFragmentByRowCol} \sqsubseteq = 1\texttt{hasToCol}$ $\texttt{DocumentFragmentByRowCol} \sqsubseteq = 1\texttt{hasFromRow}$ $\texttt{DocumentFragmentByRowCol} \sqsubseteq = 1\texttt{hasFromCol}$ $\texttt{IdentifiedThing} \sqsubseteq \leq 1\texttt{hasAuthorList}$ $\texttt{IdentifiedThing} \sqsubseteq \leq 1\texttt{hasCreationDateTime}$ $\texttt{InferenceStep} \sqsubseteq \leq 1\texttt{hasIndex}$ $\texttt{InferenceStep} \sqsubseteq \leq 1\texttt{hasSourceUsage}$ $\texttt{InferenceStep} \sqsubseteq \leq 1\texttt{hasInferenceRule}$ $\texttt{InferenceStep} \sqsubseteq \leq 1\texttt{hasInferenceEngine}$ $\texttt{InferenceStep} \sqsubseteq \leq 1\texttt{hasAntecedentList}$ $\texttt{InferenceRule} \sqsubseteq \leq 1\texttt{hasContent}$ $\texttt{Information} \sqsubseteq \leq 1\texttt{hasLanguage}$ $\texttt{Information} \sqsubseteq \leq 1\texttt{hasFormat}$ $\texttt{Information} \sqsubseteq \leq 1\texttt{hasURL}$ $\texttt{Information} \sqsubseteq \leq 1\texttt{hasPrettyString}$ $\texttt{Information} \sqsubseteq \leq 1\texttt{hasRawString}$ $\texttt{JustificationElement} \sqsubseteq \leq 1\texttt{hasCreationDateTime}$ $\texttt{Mapping} \sqsubseteq = 1\texttt{mapFrom}$ $\texttt{Mapping} \sqsubseteq = 1\texttt{mapTo}$ $\texttt{NodeSet} \sqsubseteq = 1\texttt{hasConclusion}$ $\texttt{Publication} \sqsubseteq \leq 1\texttt{hasISBN}$ $\texttt{Publication} \sqsubseteq \leq 1\texttt{hasPublicationDateTime}$ $\texttt{Question} \sqsubseteq \leq 1\texttt{hasContent}$ $\texttt{SourceUsage} \sqsubseteq = 1\texttt{hasSource}$ $\texttt{SourceUsage} \sqsubseteq = 1\texttt{hasUsageDateTime}$ $\texttt{Software} \sqsubseteq \leq 1\texttt{hasVersion}$ $\texttt{TranslationRule} \sqsubseteq \leq 1\texttt{hasToLanguage}$ $\texttt{TranslationRule} \sqsubseteq \leq 1\texttt{hasFromLanguage}$
Additional ICs	$\texttt{InferenceStep} \sqsubseteq \exists\texttt{isConsequenceOf}^{-}.\texttt{NodeSet}$, $\texttt{InferenceStep} \sqsubseteq \exists\texttt{hasIndex}.\mathrm{xsd{:}int}$, $\texttt{InferenceStep} \sqsubseteq ((\exists\texttt{hasInferenceRule}.\texttt{InferenceRule} \sqcap \exists\texttt{hasInferenceEngine}.\texttt{InferenceEngine}) \sqcup \exists\texttt{hasSourceUsage}.\texttt{SourceUsage})$, $\texttt{Information} \sqsubseteq \exists\texttt{hasRawString}.\mathrm{xsd{:}string} \sqcup \texttt{hasURL}.\mathrm{xsd{:}anyURI}$, $\texttt{NodeSet} \sqsubseteq \exists\texttt{isConsequenceOf}.\texttt{InferenceStep}$, $\texttt{Query} \sqsubseteq \exists\texttt{hasAnswer}.\texttt{NodeSet}$, $\exists\texttt{hasConclusion}^{-}.\top \sqsubseteq \exists\texttt{hasFormat}.\texttt{Format}$

Table 7.4: ICs for MLSO data

ICs from onto	$\texttt{AltitudeDependentParameter} \sqsubseteq \exists \texttt{hasCoordinate.Altitude}$, $\texttt{AnionParameter} \sqsubseteq \exists \texttt{hasElectricCharge.}\{\texttt{NegativeElectricCharge}\}$, $\texttt{AtmosphereLayer} \sqsubseteq \exists \texttt{hasSuperPhysicalDomain.Atomosphere}$, $\texttt{AuroralBoundaryIndex} \sqsubseteq \exists \texttt{hasCoordinate.Time}$, $\texttt{CationParameter} \sqsubseteq \exists \texttt{hasElectricCharge.}\{\texttt{PositiveElectricCharge}\}$, $\texttt{Dataset} \sqsubseteq \geq 1 \texttt{hasContainedParameter}$, $\texttt{Dataset} \sqsubseteq = 1 \texttt{isFromInstrument}$, $\texttt{Dataset} \sqsubseteq = 1 \texttt{isFromInstrumentOperatingMode}$, $\texttt{Deployment} \sqsubseteq = 1 \texttt{hasDeploymentId}$, $\texttt{ElectronParameter} \sqsubseteq \exists \texttt{hasElectricCharge.}\{\texttt{NegativeOneElectricCharge}\}$, $\texttt{ElectronState} \sqsubseteq \forall \texttt{hasStateParameter.ElectronParameter}$, $\texttt{GeoPlot} \sqsubseteq \exists \texttt{hasPlottedParameter.LatitudeDependentParameter}$, $\texttt{GeoPlot} \sqsubseteq \exists \texttt{hasPlottedParameter.LongitudeDependentParameter}$, $\texttt{GroundBasedInstrument} \sqsubseteq = 1 \texttt{hasLatitude}$, $\texttt{GroundBasedInstrument} \sqsubseteq = 1 \texttt{hasLongitude}$, $\texttt{HeightVsTimePlot} \sqsubseteq \exists \texttt{hasPlottedParameter.}(\texttt{AltitudeDependentParameter} \sqcap \texttt{TimeDependentParameter})$, $\texttt{IonState} \sqsubseteq \forall \texttt{hasStateParameter.CationParameter}$, $\texttt{LatitudeCircle} \sqsubseteq = 1 \texttt{hasLatitude}$, $\texttt{LatitudeDependentParameter} \sqsubseteq \exists \texttt{hasCoordinate.Latitude}$, $\texttt{LongitudeCircle} \sqsubseteq = 1 \texttt{hasLongitude}$, $\texttt{LongitudeDependentParameter} \sqsubseteq \exists \texttt{hasCoordinate.Longitude}$, $\texttt{NeutralParameter} \sqsubseteq \exists \texttt{hasElectricCharge.}\{\texttt{ZeroElectricCharge}\}$, $\texttt{NeutralState} \sqsubseteq \forall \texttt{hasStateParameter.NeutralParameter}$, $\texttt{ProtonParameter} \sqsubseteq \exists \texttt{hasElectricCharge.}\{\texttt{PositiveOneElectricCharge}\}$, $\texttt{StationaryDeployment} \sqsubseteq = 1 \texttt{hasAltitude}$, $\texttt{StationaryDeployment} \sqsubseteq = 1 \texttt{hasLatitude}$, $\texttt{StationaryDeployment} \sqsubseteq = 1 \texttt{hasLongitude}$, $\texttt{SurfaceLine} \sqsubseteq \exists \texttt{hasSuperPhysicalDomain.Surface}$, $\texttt{TimeDependentParameter} \sqsubseteq \exists \texttt{hasCoordinate.Time}$, $\texttt{TimeSeriesPlot} \sqsubseteq \exists \texttt{hasPlottedParameter.TimeDependentParameter}$,
Additional ICs	N/A

Table 7.5: ICs for CEDAR data

ICs from onto.	$\texttt{Dataset} \sqsubseteq = 1\texttt{isFromInstrument}$, $\texttt{Dataset} \sqsubseteq = 1\texttt{isFromInstrumentOperatingMode}$, $\texttt{Dataset} \sqsubseteq \geq 1\texttt{hasContainedParameter}$,
Additional ICs	$\texttt{Dataset} \sqsubseteq \exists\texttt{hasName}.\text{xsd:string}$, $\texttt{Dataset} \sqsubseteq \exists\texttt{hasDateTimeCoverage}.\texttt{DateTimeInterval}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{year}.\text{xsd:gYear}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{month}.\text{xsd:gMonth}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{day}.\text{xsd:gDay}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{hour}.\text{xsd:nonNegativeInteger}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{minute}.\text{xsd:nonNegativeInteger}$, $\texttt{DateTimeDescription} \sqsubseteq \exists\texttt{second}.\text{xsd:decimal}$, $\texttt{DateTimeInterval} \sqsubseteq \exists\texttt{hasBeginning}.\texttt{Instant}$, $\texttt{DateTimeInterval} \sqsubseteq \exists\texttt{hasEnd}.\texttt{Instant}$, $\texttt{Instant} \sqsubseteq \exists\texttt{inDateTime}.\texttt{DateTimeDescription}$, $\texttt{Instant} \sqsubseteq \exists\texttt{inXSDDateTime}.\text{xsd:dateTime}$

Table 7.6: ICs for CEDAR Time data

ICs from onto.	$\texttt{Award} \sqsubseteq = 1\texttt{hasAwardNumber}$, $\texttt{DataSet} \sqsubseteq = 1\texttt{hasDatasetName}$, $\texttt{Deployment} \sqsubseteq = 1\texttt{hasDeploymentId}$, $\texttt{DeploymentDataset} \sqsubseteq \leq 1\texttt{fromDeployment}$, $\texttt{DeploymentDatasetCollection} \sqsubseteq \forall\texttt{hasInventory}.\texttt{DeploymentDataset}$, $\texttt{Person} \sqsubseteq = 1\texttt{hasPersonFirstName}$, $\texttt{Person} \sqsubseteq = 1\texttt{hasPersonLastName}$
Additional ICs	$\texttt{DeploymentDataset} \sqsubseteq \exists\texttt{hasTimeCoverage}.\top$, $\texttt{DeploymentDataset} \sqsubseteq \exists\texttt{hasParamter}.\top$

Table 7.7: ICs for BCODMO data

ICs from onto.	**N/A**
Additional ICs	$\texttt{AbstractDataset} \sqsubseteq \exists\texttt{title}.\text{rdfs:Literal}$, $\texttt{AbstractDataset} \sqsubseteq \exists\texttt{description}.\text{rdfs:Literal}$, $\texttt{AbstractDataset} \sqsubseteq \exists\texttt{source\_agency}.\top$, $\texttt{AbstractDataset} \sqsubseteq \exists\texttt{dataset\_category}.\text{rdfs:Literal}$

Table 7.8: ICs for datagov data

Instance Data	**ICs Violated**	**Violations**
Wine	6	51
MLSO	1	53
CEDAR	6	111
CEDAR Time	3	804
BCODMO	2	2546
Datagov	4	892

Table 7.9: Results of evaluation on instance data

ICs in Databases	**OWL IC Axioms**
Not Null	`owl:minCardinality` 1
Key	`owl:maxCardinality` 1
Unique Key	`owl:maxCardinality` 1, uniqueness
Primary Key	`owl:cardinality` 1, uniqueness
Foreign Key	`owl:allValuesFrom`

Table 7.10: Mapping ICs in databases to OWL IC axioms

1U	**2U**	**3U**	**4U**	**5U**	**6U**	**7U**	**8U**	**9U**	**10U**
103K	237K	348K	494K	646K	748K	914K	1036K	1163K	1316K

Table 7.11: LUBM datasets

IC_1	$\texttt{Publication} \sqsubseteq \texttt{Books} \sqcup \texttt{JournalArticle} \sqcup \texttt{ConferencePaper} \sqcup \texttt{TechnicalReport}$,
IC_2	$\texttt{Publication} \sqsubseteq \exists\texttt{publicationDate}.\top$,
IC_3	$\texttt{ResearchGroup} \sqsubseteq \exists\texttt{reserachProject}.\top$,
IC_4	$\texttt{FullProfessor} \sqsubseteq \forall\texttt{teacherOf}.\texttt{GraduateCourse}$,
IC_5	$\texttt{AssistantProfessor} \sqsubseteq \geq 3\texttt{teacherOf}.\top$

Table 7.12: ICs for performance analysis

CHAPTER 8
Conclusions and Future Work

In this work, we have described an OWL 2 DL-based approach that we have designed and implemented for integrity constraints (ICs) on the Semantic Web. The approach enables OWL to be used as an IC language, provides services for IC validation and IC violation explanation, therefore provides a complete framework for data validation on the Semantic Web.

The standard semantics for OWL 2 DL adopts the OWA and the nUNA that is suitable for the distributed knowledge representation scenarios on the Semantic Web, where usually the knowledge about the domain comes from distributed sources and a complete knowledge about the domain cannot be assumed, but makes it difficult to use OWL for integrity constraint validation purposes. We proposed an IC semantics for OWL 2 DL that adopts the CWA and the weak UNA which enables OWL to be used as an IC language. A solution to IC validation based on conjunctive query answering is designed: an extended KB $\langle \mathcal{K}, \mathcal{C} \rangle$ is valid if and only if the conjunctive queries corresponding to $\mathcal{C}$ have empty answers. Since answers to the conjunctive queries are affected by the inferences computed by the standard OWL axioms in $\mathcal{K}$, open world reasoning and closed world constraint validation are combined on the extended KB.

We also designed a solution to explanation and repair of IC violations. This solution is based on an general approach to conjunctive query answer explanation in OWL 2 DL. To address the general problem of conjunctive query answer explanation, we first formalized the notion of justification for conjunctive query entailments, reduced the problem of conjunctive query answer explanation to justification of corresponding conjunctive query entailments, then we designed a set of algorithms for the explanation computations. Based on the approach to conjunctive query answer explanation, the explanations and repairs for IC violations can be straightforwardly computed.

We have built a prototype to implement the OWL 2 DL-based approach which accepts the instance data and the OWL IC axioms as inputs, reports the violations detected, computes the explanations for the violations, generates the repairs for the violations and applies the repairs to the instance data to fix the violations. To evaluate our approach, we have collected some instance data from the real world, selected some ICs for each dataset, and performed the IC validation. The results show that a considerable amount of ICs have been violated by a large number of individuals in the dataset. Leaving these violations in the data without repairing them and notifying the users may result in application failures. In the scenarios that data correctness and integrity is important, these violations may cause severe problems. Using our approach, it is easy to detect these issues in the data. We also conducted some performance evaluation to the validation time and justification time. It shows that our approach can be used to validate, explain, repair the datasets in reasonable time.

The possible future work includes: first, the weak UNA in the IC semantics is similar to the minimal model semantics where the equality relations are treated as congruence relations and minimized. Further research about the relationship between them, and, if possible, to generalize the IC semantics; second, the query-based IC validation approach works only when the expressivity of the extended KB is within $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$. Explore approaches of IC validation for KBs more expressive than $\langle \mathcal{SRI}, \mathcal{SROIQ} \rangle$ or $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$: given a KB K with the expressivity $\langle \mathcal{SROIQ}, \mathcal{SROIQ} \rangle$, decompose it into two KBs, i.e., a $\langle \mathcal{SROIQ}, \mathcal{SROI} \rangle$ KB A which is supported by the solution, and a $\langle \mathcal{SROIQ}, \mathcal{Q} \rangle$ KB B which needs alternative solutions; third, to optimize IC violation justification: (1) Optimize query answer justification algorithms. In the presence of negation of conjunction, find heuristics to decrease the searching space of possible combinations; (2) Optimize justification minimization. In the process of minimization, DL reasoning is invoked to remove redundant axioms. Find solutions to improve the performance. One possible solution is incremental reasoning where the DL reasoners do not need to repeat all reasoning tasks when the KBs are updated; fourth, the external DL axiom entailment justification service called by the IC violation justification does not scale well when the dataset size increases. It can be replaced by alternative axiom entailment justification services if such services exist.

APPENDIX A
Proof of Lemma 4

Proof The proof is constructed inductively on the structure of concept C which is a $\mathcal{SROIQ}$ concept. Any $\mathcal{SROIQ}$ concept must belong to the five cases: C_a, $\neg C$, $C_1 \sqcap C_2$, $\geq nR.C$, $\exists R.\text{Self}$. The complete proof is as follows.

Base Case: $\mathcal{K} \models_{IC} C_a(a)$ iff $\mathcal{K} \models \mathcal{T}_c(C_a, a)$, $C_a \in N_C$.

$\mathcal{K} \models_{IC} C_a(a)$

$\Longleftrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (C_a)^{\mathcal{I},\mathcal{U}}, \mathcal{U} = Mod_{ME}(\mathcal{K})$

$\Longleftrightarrow \forall \mathcal{I} \in Mod_{ME}(\mathcal{K}), a^{\mathcal{I}} \in (C_a)^{\mathcal{I}}[a^{\mathcal{I},\mathcal{U}} = a^{\mathcal{I}}, (C_a)^{\mathcal{I},\mathcal{U}} \subseteq (C_a)^{\mathcal{I}}]$

$\Longleftrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), a^{\mathcal{I}} \in (C_a)^{\mathcal{I}}[\texttt{by Lemma 2}]$

$\Longleftrightarrow \mathcal{K} \models C_a(a)$

$\Longleftrightarrow \mathcal{K} \models \mathcal{T}_c(C_a, a)[\texttt{by definition of } \mathcal{T}_c(C_a, x)]$

Induction Steps: We now prove by induction that for any complex SROIQ concept C, $\mathcal{K} \models_{IC} C(a)$ iff $\mathcal{K} \models \mathcal{T}_c(C, a)$. As the basis of the proof, assume that $\mathcal{K} \models_{IC} C_{(i)}(a)$ iff $K \models \mathcal{T}_c(C_{(i)}, a)$, where subscript i=1,2 is optional. We enumerate all possible cases of SROIQ concepts as follows.

Case 1: $\mathcal{K} \models_{IC} (\neg C)(a)$ iff $\mathcal{K} \models \mathcal{T}_c(\neg C, a)$

$\mathcal{K} \models_{IC} (\neg C)(a)$

$\Longleftrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (\neg C)^{\mathcal{I},\mathcal{U}} = (N_I)^{\mathcal{I}} \setminus C^{\mathcal{I},\mathcal{U}}$

$\Longleftrightarrow \exists \mathcal{J} \in \mathcal{U}, a^{\mathcal{J}} \notin C^{\mathcal{J}}$

$\Longleftrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (C)^{\mathcal{I},\mathcal{U}}$

$\Longleftrightarrow \mathcal{K} \not\models_{IC} C(a)$

$\Longleftrightarrow \mathcal{K} \not\models \mathcal{T}_c(C, a)[\texttt{by assumption}]$

$\Longleftrightarrow \mathcal{K} \models \textbf{not}\, \mathcal{T}_c(C, a)$

$\Longleftrightarrow \mathcal{K} \models \mathcal{T}_c(\neg C, a)[\texttt{by definition of } \mathcal{T}_c(\neg C, x)]$

Case 2: $\mathcal{K} \models_{IC} (C_1 \sqcap C_2)(a)$ iff $\mathcal{K} \models \mathcal{T}_c(C_1 \sqcap C_2, a)$

$\mathcal{K} \models_{IC} (C_1 \sqcap C_2)(a)$

$\Longleftrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (C_1 \sqcap C_2)^{\mathcal{I},\mathcal{U}} = {C_1}^{\mathcal{I},\mathcal{U}} \cap {C_2}^{\mathcal{I},\mathcal{U}}$

$\Longleftrightarrow \mathcal{K} \models_{IC} C_1(a), \mathcal{K} \models_{IC} C_2(a)$

$\Longleftrightarrow \mathcal{K} \models \mathcal{T}_c(C_1, a), \mathcal{K} \models \mathcal{T}_c(C_2, a)$[`by assumption`]

$\Longleftrightarrow \mathcal{K} \models \mathcal{T}_c(C_1 \sqcap C_2, a)$[`by definition of` $\mathcal{T}_c(C_1 \sqcap C_2, x)$]

Case 3: $\mathcal{K} \models_{IC} (\geq nR.C)(a)$ iff $\mathcal{K} \models \mathcal{T}_c(\geq nR.C, a)$

$\mathcal{K} \models_{IC} (\geq nR.C)(a)$

$\Longleftrightarrow \forall \mathcal{I} \in \mathcal{U}, \#\{y^{\mathcal{I}} \mid \langle a^{\mathcal{I},\mathcal{U}}, y^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}, y^{\mathcal{I},\mathcal{U}} \in C^{\mathcal{I},\mathcal{U}}\} \geq n$

$\Longleftrightarrow$ [`by Lemma 3`]

$\exists y_i \in N_I (1 \leq i \leq n)$ s.t.

$\forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, (y_i)^{\mathcal{I}} \rangle \in R^{\mathcal{I}}, [a^{\mathcal{I},\mathcal{U}} = a^{\mathcal{I}}, (y_i)^{\mathcal{I},\mathcal{U}} = (y_i)^{\mathcal{I}}, (R)^{\mathcal{I},\mathcal{U}} \subseteq (R)^{\mathcal{I}}]$

$\forall \mathcal{I} \in \mathcal{U}, (y_i)^{\mathcal{I},\mathcal{U}} \in C^{\mathcal{I},\mathcal{U}}, (y_i)^{\mathcal{I}} \neq (y_j)^{\mathcal{I}} (1 \leq i < j \leq n)$

$\Longleftrightarrow \exists y_i \in N_I (1 \leq i \leq n)$ s.t.

$\forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, (y_i)^{\mathcal{I}} \rangle \in R^{\mathcal{I}},$ [`by Lemma 2`]

$\mathcal{K} \models_{IC} C(y_i), \mathcal{K} \models \textbf{not}\,(y_i = y_j)(1 \leq i < j \leq n)$

$\Longleftrightarrow \exists y_i \in N_I (1 \leq i \leq n)$ s.t.

$\mathcal{K} \models R(a, y_i), \mathcal{K} \models \mathcal{T}_c(C, y_i),$ [`by assumption`]

$\mathcal{K} \models \textbf{not}\,(y_i = y_j)(1 \leq i < j \leq n)$

$$\Longleftrightarrow \mathcal{K} \models \bigwedge_{1 \leq i \leq n} (R(a, y_i) \wedge \mathcal{T}_c(C, y_i)) \bigwedge_{1 \leq i < j \leq n} \textbf{not}\,(y_i = y_j)$$

$\Longleftrightarrow \mathcal{K} \models \mathcal{T}_c(\geq nR.C, a)$

Case 4: $\mathcal{K} \models_{IC} (\exists R.\text{Self})(a)$ iff $\mathcal{K} \models \mathcal{T}_c(\exists R.\text{Self}, a)$

$\mathcal{K} \models_{IC} (\exists R.\text{Self})(a)$

$\iff \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (\exists R.\text{Self})^{\mathcal{I},\mathcal{U}}$

$\iff \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, a^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\iff \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, a^{\mathcal{I}} \rangle \in R^{\mathcal{I}}[a^{\mathcal{I},\mathcal{U}} = a^{\mathcal{I}}, (R)^{\mathcal{I},\mathcal{U}} \subseteq (R)^{\mathcal{I}}]$

$\iff \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, a^{\mathcal{I}} \rangle \in R^{\mathcal{I}}[\texttt{by Lemma 2}]$

$\iff \mathcal{K} \models R(a, a)$

$\iff \mathcal{K} \models \mathcal{T}_c(\exists R.\text{Self}, a)[\texttt{by definition of } \mathcal{T}_c(\exists R.\text{Self}, x)]$

APPENDIX B
Proof of Theorem 1

Proof We enumerate all possible cases of α, which is a $\mathcal{SROIQ}$ axiom, and for each case we prove the conclusion is true. Any $\mathcal{SROIQ}$ axiom must belong to these cases: $C_1 \sqsubseteq C_2$, $R_1 \sqsubseteq R_2$, $R_1 \ldots R_n \sqsubseteq R$, $\texttt{Ref}(R)$, $\texttt{Irr}(R)$, $\texttt{Dis}(R_1, R_2)$, $C(a)$, $R(a,b)$, $\neg R(a,b)$, $a = b$, $a \neq b$. The complete proof is as follows.

Case 1: $\mathcal{K} \models_{IC} C_1 \sqsubseteq C_2$ iff $\mathcal{K} \not\models \mathcal{T}(C_1 \sqsubseteq C_2)$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} C_1 \sqsubseteq C_2$ then $\mathcal{K} \not\models \mathcal{T}(C_1 \sqsubseteq C_2)$.

Assume to the contrary

$$\mathcal{K} \models \mathcal{T}(C_1 \sqsubseteq C_2)$$

$$\Longrightarrow \mathcal{K} \models \mathcal{T}_c(C_1, x) \wedge \textbf{not}\, \mathcal{T}_c(C_2, x)[\texttt{by definition of } \mathcal{T}(C_1 \sqsubseteq C_2)]$$

$$\Longrightarrow \exists \sigma : x \to a, \mathcal{K} \models \mathcal{T}_c(C_1, a), \mathcal{K} \not\models \mathcal{T}_c(C_2, a)$$

$$\Longrightarrow \mathcal{K} \models_{IC} C_1(a), \mathcal{K} \not\models_{IC} C_2(a)[\texttt{by Lemma 4}]$$

$$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \in (C_1)^{\mathcal{I},\mathcal{U}}, \exists \mathcal{J} \in \mathcal{U}, a^{\mathcal{J},\mathcal{U}} \notin (C_2)^{\mathcal{J},\mathcal{U}}$$

$$\Longrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, (C_1)^{\mathcal{I},\mathcal{U}} \subseteq (C_2)^{\mathcal{I},\mathcal{U}}$$

$$\Longrightarrow \mathcal{K} \not\models_{IC} C_1 \sqsubseteq C_2$$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(C_1 \sqsubseteq C_2)$ then $\mathcal{K} \models_{IC} C_1 \sqsubseteq C_2$.

$\mathcal{K} \not\models \mathcal{T}(C_1 \sqsubseteq C_2)$

$\Longrightarrow \mathcal{K} \not\models \mathcal{T}_c(C_1, x) \wedge \textbf{not}\, \mathcal{T}_c(C_2, x)[\texttt{by definition of } \mathcal{T}(C_1 \sqsubseteq C_2)]$

$\Longrightarrow \nexists \sigma : x \rightarrow a, K \models \mathcal{T}_c(C_1, a), \mathcal{K} \not\models \mathcal{T}_c(C_2, a)$

$\Longrightarrow \forall \sigma : x \rightarrow a, \texttt{if } \mathcal{K} \models \mathcal{T}_c(C_1, a), \texttt{then } \mathcal{K} \models \mathcal{T}_c(C_2, a)$

$\Longrightarrow \forall x \in N_I, \texttt{if } \mathcal{K} \models_{IC} C_1(x), \texttt{then } \mathcal{K} \models_{IC} C_2(x)[\texttt{by Lemma 4}]$

$\Longrightarrow \forall x \in N_I, \texttt{if } \forall \mathcal{I} \in \mathcal{U}, x^{\mathcal{I},\mathcal{U}} \in (C_1)^{\mathcal{I},\mathcal{U}}, \texttt{then } \forall \mathcal{I} \in \mathcal{U}, x^{\mathcal{I},\mathcal{U}} \in (C_2)^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, (C_1)^{\mathcal{I},\mathcal{U}} \subseteq (C_2)^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} C_1 \sqsubseteq C_2$

Case 2: $\mathcal{K} \models_{IC} R_1 \sqsubseteq R_2$ iff $\mathcal{K} \not\models \mathcal{T}(R_1 \sqsubseteq R_2)$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} R_1 \sqsubseteq R_2$ then $\mathcal{K} \not\models \mathcal{T}(R_1 \sqsubseteq R_2)$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(R_1 \sqsubseteq R_2)$

$\Longrightarrow \mathcal{K} \models R_1(x, y) \wedge \textbf{not}\, R_2(x, y)[\texttt{by definition of } \mathcal{T}(R_1 \sqsubseteq R_2)]$

$\Longrightarrow \exists \sigma : x \rightarrow a, y \rightarrow b, \mathcal{K} \models R_1(a, b), \mathcal{K} \not\models R_2(a, b)$

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R_1^{\mathcal{I}}, \exists \mathcal{J} \in Mod(\mathcal{K}), \langle a^{\mathcal{J}}, b^{\mathcal{J}} \rangle \notin R_2^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R_1^{\mathcal{I}}, \exists \mathcal{J}' \in \mathcal{U}, \langle a^{\mathcal{J}'}, b^{\mathcal{J}'} \rangle \notin R_2^{\mathcal{J}'}[\texttt{by Lemma 2}]$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in R_1^{\mathcal{I},\mathcal{U}}, \exists \mathcal{J}' \in \mathcal{U}, \langle a^{\mathcal{J}',\mathcal{U}}, b^{\mathcal{J}',\mathcal{U}} \rangle \notin R_2^{\mathcal{J}',\mathcal{U}}$

$\Longrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, (R_1)^{\mathcal{I},\mathcal{U}} \subseteq (R_2)^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} R_1 \sqsubseteq R_2$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(R_1 \sqsubseteq R_2)$ then $\mathcal{K} \models_{IC} R_1 \sqsubseteq R_2$.

$\mathcal{K} \not\models \mathcal{T}(R_1 \sqsubseteq R_2)$

$\Longrightarrow \mathcal{K} \not\models R_1(x,y) \wedge \textbf{not}\, R_2(x,y)[\texttt{by definition of } \mathcal{T}(R_1 \sqsubseteq R_2)]$

$\Longrightarrow \nexists\sigma : x \to a, y \to b, K \models R_1(a,b), \mathcal{K} \not\models R_2(a,b)$

$\Longrightarrow \forall\sigma : x \to a, y \to b, \texttt{if } \mathcal{K} \models R_1(a,b), \texttt{then } \mathcal{K} \models R_2(a,b)$

$\Longrightarrow \forall x,y \in N_I, \texttt{if } \forall\mathcal{I} \in Mod(\mathcal{K}), \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R_1^{\mathcal{I}}\texttt{then } \forall\mathcal{I} \in Mod(\mathcal{K}), \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R_2^{\mathcal{I}}$

$\Longrightarrow \forall x,y \in N_I, \texttt{if } \forall\mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R_1^{\mathcal{I}}\texttt{then } \forall\mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \in R_2^{\mathcal{I}}[\texttt{by Lemma 2}]$

$\Longrightarrow \forall x,y \in N_I, \texttt{if } \forall\mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I},\mathcal{U}}, y^{\mathcal{I},\mathcal{U}}\rangle \in (R_1)^{\mathcal{I},\mathcal{U}}, \texttt{then } \forall\mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I},\mathcal{U}}, y^{\mathcal{I},\mathcal{U}}\rangle \in (R_2)^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \forall\mathcal{I} \in \mathcal{U}, (R_1)^{\mathcal{I},\mathcal{U}} \subseteq (R_2)^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} R_1 \sqsubseteq R_2$

Case 3: $\mathcal{K} \models_{IC} R_1 \ldots R_n \sqsubseteq R$ iff $\mathcal{K} \not\models \mathcal{T}(R_1 \ldots R_n \sqsubseteq R)$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} R_1 \ldots R_n \sqsubseteq R$ then $\mathcal{K} \not\models \mathcal{T}(R_1 \ldots R_n \sqsubseteq R)$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(R_1 \ldots R_n \sqsubseteq R)$

$\Longrightarrow \mathcal{K} \models R_1(x,y_1) \wedge \ldots R_n(y_{n-1},y_n) \wedge \textbf{not}\, R(x,y_n)[\texttt{by definition of } \mathcal{T}(R_1 \ldots R_n \sqsubseteq R)]$

$\Longrightarrow \exists\sigma : x \to a, y_1 \to b_1, \ldots, y_n \to b_n, s.t.$

$\mathcal{K} \models R_1(a,b_1), \ldots, \quad \mathcal{K} \models R_n(b_{n-1},b_n), \mathcal{K} \not\models R(a,b_n)$

$\Longrightarrow \forall\mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, {b_1}^{\mathcal{I}}\rangle \in R_1^{\mathcal{I}}, \ldots, \langle {b_{n-1}}^{\mathcal{I}}, {b_n}^{\mathcal{I}}\rangle \in R_n^{\mathcal{I}},$

$\exists\mathcal{J} \in Mod(\mathcal{K}), \langle a^{\mathcal{J}}, {b_n}^{\mathcal{J}}\rangle \notin R^{\mathcal{I}}$

$\Longrightarrow \forall\mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, {b_1}^{\mathcal{I}}\rangle \in R_1^{\mathcal{I}}, \ldots, \langle {b_{n-1}}^{\mathcal{I}}, {b_n}^{\mathcal{I}}\rangle \in R_n^{\mathcal{I}},$

$\exists\mathcal{J}' \in \mathcal{U}, \langle a^{\mathcal{J}'}, {b_n}^{\mathcal{J}'}\rangle \notin R^{\mathcal{J}'}[\texttt{by Lemma 2}]$

$\Longrightarrow \forall\mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, {b_1}^{\mathcal{I},\mathcal{U}}\rangle \in R_1^{\mathcal{I},\mathcal{U}}, \ldots, \langle {b_{n-1}}^{\mathcal{I},\mathcal{U}}, {b_n}^{\mathcal{I},\mathcal{U}}\rangle \in R_n^{\mathcal{I},\mathcal{U}},$

$\exists\mathcal{J}' \in \mathcal{U}, \langle a^{\mathcal{J}',\mathcal{U}}, {b_n}^{\mathcal{J}',\mathcal{U}}\rangle \notin R^{\mathcal{J}',\mathcal{U}}$

$\Longrightarrow \texttt{It is false that}\forall\mathcal{I} \in \mathcal{U}, R_1^{\mathcal{I},\mathcal{U}} \circ \ldots \circ R_n^{\mathcal{I},\mathcal{U}} \subseteq R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} R_1 \ldots R_n \sqsubseteq R$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(R_1 \dots R_n \sqsubseteq R)$ then $\mathcal{K} \models_{IC} R_1 \dots R_n \sqsubseteq R$.

$\mathcal{K} \not\models \mathcal{T}(R_1 \dots R_n \sqsubseteq R)$

$\Longrightarrow \mathcal{K} \not\models R_1(x, y_1) \wedge \dots R_n(y_{n-1}, y_n) \wedge \textbf{not}\, R(x, y_n)$[`by definition of` $\mathcal{T}(R_1 \dots R_n \sqsubseteq R)$]

$\Longrightarrow \nexists \sigma : x \to a, y_1 \to b_1, \dots, y_n \to b_n, s.t.$

$K \models R_1(a, b), \dots, K \models R_n(b_{n-1}, b_n), \mathcal{K} \not\models R(a, b_n)$

$\Longrightarrow \forall \sigma : x \to a, y_1 \to b_1, \dots, y_n \to b_n,$

`if` $\mathcal{K} \models R_1(a, b), \dots, K \models R_n(b_{n-1}, b_n),$ `then` $\mathcal{K} \models R(a, b_n)$

$\Longrightarrow \forall x, y_1, \dots, y_n \in N_I,$

`if` $\forall \mathcal{I} \in Mod(\mathcal{K}), \langle x^{\mathcal{I}}, {y_1}^{\mathcal{I}} \rangle \in R_1^{\mathcal{I}} \dots, \langle {y_{n-1}}^{\mathcal{I}}, {y_n}^{\mathcal{I}} \rangle \in R_n^{\mathcal{I}}$

`then` $\forall \mathcal{I} \in Mod(\mathcal{K}), \langle x^{\mathcal{I}}, {y_n}^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

$\Longrightarrow \forall x, y_1, \dots, y_n \in N_I,$

`if` $\forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I}}, {y_1}^{\mathcal{I}} \rangle \in R_1^{\mathcal{I}} \dots, \langle {y_{n-1}}^{\mathcal{I}}, {y_n}^{\mathcal{I}} \rangle \in R_n^{\mathcal{I}}$

`then` $\forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I}}, {y_n}^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall x, y_1, \dots, y_n \in N_I,$

`if` $\forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I},\mathcal{U}}, {y_1}^{\mathcal{I},\mathcal{U}} \rangle \in R_1^{\mathcal{I},\mathcal{U}} \dots, \langle {y_{n-1}}^{\mathcal{I},\mathcal{U}}, {y_n}^{\mathcal{I},\mathcal{U}} \rangle \in R_n^{\mathcal{I},\mathcal{U}}$

`then` $\forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I},\mathcal{U}}, {y_n}^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, R_1^{\mathcal{I},\mathcal{U}} \circ \dots \circ R_n^{\mathcal{I},\mathcal{U}} \subseteq R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} R_1 \dots R_n \sqsubseteq R$

Case 4: $\mathcal{K} \models_{IC}$ `Ref`(R) iff $\mathcal{K} \not\models \mathcal{T}$(`Ref`(R))

($\Longrightarrow$): if $\mathcal{K} \models_{IC}$ `Ref`(R) then $\mathcal{K} \not\models \mathcal{T}$(`Ref`(R)).

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(\texttt{Ref}(R))$

$\Longrightarrow \mathcal{K} \models \textbf{not}\, R(x,x)[\texttt{by definition of } \mathcal{T}(\texttt{Ref}(R))]$

$\Longrightarrow \forall x \in N_I, \exists \mathcal{I} \in Mod(\mathcal{K}), s.t. \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}.$

$\Longrightarrow \forall x \in N_I, \exists \mathcal{I}' \in \mathcal{U}, s.t. \langle x^{\mathcal{I}'}, x^{\mathcal{I}'} \rangle \notin R^{\mathcal{I}'}.[\texttt{by Lemma 2}]$

$\Longrightarrow \forall x \in N_I, \exists \mathcal{I}' \in \mathcal{U}, s.t. \langle x^{\mathcal{I}',\mathcal{U}}, x^{\mathcal{I}',\mathcal{U}} \rangle \notin R^{\mathcal{I}',\mathcal{U}}.$

$\Longrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, \forall x \in N_I, \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} \texttt{Ref}(R)$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(\texttt{Ref}(\text{R}))$ then $\mathcal{K} \models_{IC} \texttt{Ref}(\text{R})$.

$\mathcal{K} \not\models \mathcal{T}(\texttt{Ref}(R))$

$\Longrightarrow \mathcal{K} \not\models \textbf{not}\, R(x,x)[\texttt{by definition of } \mathcal{T}(\texttt{Ref}(R))]$

$\Longrightarrow \nexists \sigma : x \to a, s.t. \mathcal{K} \models \textbf{not}\, R(a,a)$

$\Longrightarrow \forall x \in N_I, \mathcal{K} \models R(x,x)$

$\Longrightarrow \forall x \in N_I, \forall \mathcal{I} \in Mod(\mathcal{K}), \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

$\Longrightarrow \forall x \in N_I, \forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \in R^{\mathcal{I}}[\texttt{by Lemma 2}]$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \forall x \in N_I, \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} \texttt{Ref}(R)$

Case 5: $\mathcal{K} \models_{IC} \texttt{Irr}(\text{R})$ iff $\mathcal{K} \not\models \mathcal{T}(\texttt{Irr}(\text{R}))$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} \texttt{Irr}(\text{R})$ then $\mathcal{K} \not\models \mathcal{T}(\texttt{Irr}(\text{R}))$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(\texttt{Irr}(R))$

$\Longrightarrow \mathcal{K} \models R(x,x)[\texttt{by definition of } \mathcal{T}(\texttt{Irr}(R))]$

$\Longrightarrow \exists \sigma : x \to a, s.t. \mathcal{K} \models R(a,a)$

$\Longrightarrow \exists a \in N_I, s.t. \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, a^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

$\Longrightarrow \exists a \in N_I, s.t. \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, a^{\mathcal{I}} \rangle \in R^{\mathcal{I}}[\texttt{by Lemma 2}]$

$\Longrightarrow \exists a \in N_I, s.t. \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, a^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, \forall x \in N_I, \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} \texttt{Irr}(R)$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(\texttt{Irr}(\text{R}))$ then $\mathcal{K} \models_{IC} \texttt{Irr}(\text{R})$.

$\mathcal{K} \not\models \mathcal{T}(\texttt{Irr}(R))$

$\Longrightarrow \mathcal{K} \not\models R(x,x)[\texttt{by definition of } \mathcal{T}(\texttt{Irr}(R))]$

$\Longrightarrow \forall x \in N_I, \exists \mathcal{I} \in Mod(\mathcal{K}), s.t. \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}$

$\Longrightarrow \forall x \in N_I, \exists \mathcal{I}' \in \mathcal{U}, s.t. \langle x^{\mathcal{I}'}, x^{\mathcal{I}'} \rangle \notin R^{\mathcal{I}'}[\texttt{by Lemma 2}]$

$\Longrightarrow \forall x \in N_I, \forall \mathcal{I} \in \mathcal{U}, \langle x^{\mathcal{I},\mathcal{U}}, x^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} \texttt{Irr}(R)$

Case 6: $\mathcal{K} \models_{IC} \texttt{Dis}(R_1, R_2)$ iff $\mathcal{K} \not\models \mathcal{T}(\texttt{Dis}(R_1, R_2))$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} \texttt{Dis}(R_1, R_2)$ then $\mathcal{K} \not\models \mathcal{T}(\texttt{Dis}(R_1, R_2))$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(\texttt{Dis}(R_1, R_2))$

$\Longrightarrow \mathcal{K} \models R_1(x,y) \wedge R_2(x,y)$[`by definition of` $\mathcal{T}(\texttt{Dis}(R_1, R_2))$]

$\Longrightarrow \exists \sigma : x \to a, y \to b, s.t. \mathcal{K} \models R_1(a,b), \mathcal{K} \models R_2(a,b)$

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_1}^{\mathcal{I}}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_2}^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_1}^{\mathcal{I}}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_2}^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in {R_1}^{\mathcal{I},\mathcal{U}}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in {R_2}^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in {R_1}^{\mathcal{I},\mathcal{U}} \cap {R_2}^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow$ `It is false that`$\forall \mathcal{I} \in \mathcal{U}, R_1^{\mathcal{I},\mathcal{U}} \cap R_2^{\mathcal{I},\mathcal{U}} = \emptyset$

$\Longrightarrow \mathcal{K} \not\models_{IC} \texttt{Dis}(R_1, R_2)$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(\texttt{Dis}(R_1, R_2))$ then $\mathcal{K} \models_{IC} \texttt{Dis}(R_1, R_2)$.

$\mathcal{K} \not\models \mathcal{T}(\texttt{Dis}(R_1, R_2))$

$\Longrightarrow \mathcal{K} \not\models R_1(x,y) \wedge R_2(x,y)$[`by definition of` $\mathcal{T}(\texttt{Dis}(R_1, R_2))$]

$\Longrightarrow \nexists a, b \in N_I, s.t. \mathcal{K} \models R_1(a,b), \mathcal{K} \models R_2(a,b)$

$\Longrightarrow \nexists a, b \in N_I, s.t. \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_1}^{\mathcal{I}}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_2}^{\mathcal{I}}$

$\Longrightarrow \nexists a, b \in N_I, s.t. \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_1}^{\mathcal{I}}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in {R_2}^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \nexists a, b \in N_I, s.t. \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R_1^{\mathcal{I},\mathcal{U}} \cap R_2^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, R_1^{\mathcal{I},\mathcal{U}} \cap R_2^{\mathcal{I},\mathcal{U}} = \emptyset$

$\Longrightarrow \mathcal{K} \models_{IC} \texttt{Dis}(R_1, R_2)$

Case 7: $\mathcal{K} \models_{IC} C(a)$ iff $\mathcal{K} \not\models \mathcal{T}(C(a))$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} C(a)$ then $\mathcal{K} \not\models \mathcal{T}(C(a))$.

Assume to the contrary

$$\begin{aligned}
&\mathcal{K} \models \mathcal{T}(C(a)) \\
&\Longrightarrow \mathcal{K} \models \textbf{not}\, \mathcal{T}_c(C,a)[\texttt{by definition of } \mathcal{T}(C(a))] \\
&\Longrightarrow \mathcal{K} \not\models \mathcal{T}_c(C,a) \\
&\Longrightarrow \mathcal{K} \not\models_{IC} C(a)[\texttt{by Lemma 4}]
\end{aligned}$$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(C(a))$ then $\mathcal{K} \models_{IC} C(a)$.

$$\begin{aligned}
&\mathcal{K} \not\models \mathcal{T}(C(a)) \\
&\Longrightarrow \mathcal{K} \not\models \textbf{not}\, \mathcal{T}_c(C,a)[\texttt{by definition of } \mathcal{T}(C(a))] \\
&\Longrightarrow \mathcal{K} \models \mathcal{T}_c(C,a) \\
&\Longrightarrow \mathcal{K} \models_{IC} C(a)[\texttt{by Lemma 4}]
\end{aligned}$$

Case 8: $\mathcal{K} \models_{IC} R(a,b)$ iff $\mathcal{K} \not\models \mathcal{T}(R(a,b))$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} R(a,b)$ then $\mathcal{K} \not\models \mathcal{T}(R(a,b))$.

Assume to the contrary

$$\begin{aligned}
&\mathcal{K} \models \mathcal{T}(R(a,b)) \\
&\Longrightarrow \mathcal{K} \models \textbf{not}\, R(a,b)[\texttt{by definition of } \mathcal{T}(R(a,b))] \\
&\Longrightarrow \exists \mathcal{I} \in Mod(\mathcal{K}), s.t. \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}} \\
&\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, s.t. \langle a^{\mathcal{I}'}, b^{\mathcal{I}'} \rangle \notin R^{\mathcal{I}'}[\texttt{by Lemma 2}] \\
&\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, s.t. \langle a^{\mathcal{I}',\mathcal{U}}, b^{\mathcal{I}',\mathcal{U}} \rangle \notin R^{\mathcal{I}',\mathcal{U}} \\
&\Longrightarrow \texttt{It is false that} \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}} \\
&\Longrightarrow \mathcal{K} \not\models_{IC} R(a,b)
\end{aligned}$$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(R(a,b))$ then $\mathcal{K} \models_{IC} R(a,b)$.

$\mathcal{K} \not\models \mathcal{T}(R(a,b))$

$\Longrightarrow \mathcal{K} \not\models \textbf{not}\, R(a,b)$[`by definition of` $\mathcal{T}(R(a,b))$]

$\Longrightarrow \mathcal{K} \models R(a,b)$

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} R(a,b)$

Case 9: $\mathcal{K} \models_{IC} \neg R(a,b)$ iff $\mathcal{K} \not\models \mathcal{T}(\neg R(a,b))$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} \neg R(a,b)$ then $\mathcal{K} \not\models \mathcal{T}(\neg R(a,b))$.
Assume to the contrary

$\mathcal{K} \models \mathcal{T}(\neg R(a,b))$

$\Longrightarrow \mathcal{K} \models R(a,b)$[`by definition of` $\mathcal{T}(\neg R(a,b))$]

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \in R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow$ `It is false that`$\forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} \neg R(a,b)$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(\neg R(a,b))$ then $\mathcal{K} \models_{IC} \neg R(a,b)$.

$\mathcal{K} \not\models \mathcal{T}(\neg R(a,b))$

$\Longrightarrow \mathcal{K} \not\models R(a,b)$[`by definition of` $\mathcal{T}(\neg R(a,b))$]

$\Longrightarrow \exists \mathcal{I} \in Mod(\mathcal{K}), \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}$

$\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, \langle a^{\mathcal{I}'}, b^{\mathcal{I}'} \rangle \notin R^{\mathcal{I}'}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, \langle a^{\mathcal{I},\mathcal{U}}, b^{\mathcal{I},\mathcal{U}} \rangle \notin R^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} \neg R(a,b)$

Case 10: $\mathcal{K} \models_{IC} a = b$ iff $\mathcal{K} \not\models \mathcal{T}(a = b)$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} a = b$ then $\mathcal{K} \not\models \mathcal{T}(a = b)$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(a = b)$

$\Longrightarrow \mathcal{K} \models$ **not** $(a = b)$[`by definition of` $\mathcal{T}(a = b)$]

$\Longrightarrow \exists \mathcal{I} \in Mod(\mathcal{K}), a^{\mathcal{I}} \neq b^{\mathcal{I}}$

$\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, a^{\mathcal{I}'} \neq b^{\mathcal{I}'}$[`by Lemma 2`]

$\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, a^{\mathcal{I}',\mathcal{U}} \neq b^{\mathcal{I}',\mathcal{U}}$

$\Longrightarrow$ `It is false that`$\forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} = b^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} a = b$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(a = b)$ then $\mathcal{K} \models_{IC} a = b$.

$\mathcal{K} \not\models \mathcal{T}(a = b)$

$\Longrightarrow \mathcal{K} \not\models \textbf{not}\,(a = b)$[`by definition of` $\mathcal{T}(a = b)$]

$\Longrightarrow \mathcal{K} \models (a = b)$

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), a^{\mathcal{I}} = b^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I}} = b^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} = b^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} a = b$

Case 11: $\mathcal{K} \models_{IC} a \neq b$ iff $\mathcal{K} \not\models \mathcal{T}(a \neq b)$

($\Longrightarrow$): if $\mathcal{K} \models_{IC} a \neq b$ then $\mathcal{K} \not\models \mathcal{T}(a \neq b)$.

Assume to the contrary

$\mathcal{K} \models \mathcal{T}(a \neq b)$

$\Longrightarrow \mathcal{K} \models (a = b)$[`by definition of` $\mathcal{T}(a \neq b)$]

$\Longrightarrow \forall \mathcal{I} \in Mod(\mathcal{K}), a^{\mathcal{I}} = b^{\mathcal{I}}$

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I}} = b^{\mathcal{I}}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} = b^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow$ `It is false that`$\forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \neq b^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \not\models_{IC} a \neq b$

Which is a contradiction.

($\Longleftarrow$): if $\mathcal{K} \not\models \mathcal{T}(a \neq b)$ then $\mathcal{K} \models_{IC} a \neq b$.

$\mathcal{K} \not\models \mathcal{T}(a \neq b)$

$\Longrightarrow \mathcal{K} \not\models (a = b)$[`by definition of` $\mathcal{T}(a \neq b)$]

$\Longrightarrow \exists \mathcal{I} \in Mod(\mathcal{K}), a^{\mathcal{I}} \neq b^{\mathcal{I}}$

$\Longrightarrow \exists \mathcal{I}' \in \mathcal{U}, a^{\mathcal{I}'} \neq b^{\mathcal{I}'}$[`by Lemma 2`]

$\Longrightarrow \forall \mathcal{I} \in \mathcal{U}, a^{\mathcal{I},\mathcal{U}} \neq b^{\mathcal{I},\mathcal{U}}$

$\Longrightarrow \mathcal{K} \models_{IC} a \neq b$

APPENDIX C
Proof of Theorem 2

Proof To prove Theorem 2 is to prove Alg 3 - 7 together return all justifications for the query entailments $\mathcal{K} \models^{\sigma} Q(\bar{x})$. The query entailment $\mathcal{K} \models^{\sigma} Q(\bar{x})$ holds if and only if the evaluation of query body $\sigma(Q)$ is true, which is justified by Alg 4 - 7 together. Alg 4, 5, 6 together recursively break down the given query body into atomic query atoms, justify each atomic query atom, and combine them together via Alg 7. Therefore, to prove Theorem 2, we just need to enumerate the possible cases of query $Q(\bar{x})$ and for each case to prove: (1) soundness: the results returned by Alg 4 - 7 as positive (resp. negative) justifications satisfy all the conditions 1(abc) (resp. 2(abcde)) in the justification definition (or are empty); (2) completeness: all results satisfy the conditions 1(abc) (resp. 2(abcde)) in the justification definition are returned by Alg 4 - 7 as positive (resp. negative) justifications. The details of the proof are presented in Appendix C.

Case 1: $Q \leftarrow q$

Soundness: In this case, the external algorithm JustifyAxiomEntailment is called to compute all justifications, i.e., T, for the entailment of axiom q and for each $t \in T$ a justification $\mathcal{J} = \langle \mathcal{J}_{+}, \mathcal{J}_{-} \rangle = \langle t, \emptyset \rangle$ is generated. By the definition of axiom entailment justification given in Section 6.1.1, $t \subseteq \mathcal{K}$, $t \models q$, and $\forall t' \subset t, t' \not\models q$, therefore t satisfies conditions 1(abc) and t is a positive justification ($\mathcal{J}_{+}$). Also due to the monotonic property of DLs, the entailment of axiom q by $\mathcal{K}$ would always be true no matter what axioms are added to $\mathcal{K}$. That is, there are no non-empty set of axioms satisfy conditions 2(abcde) therefore the negative justification ($\mathcal{J}_{-}$) is empty.

Completeness: Any set of axioms satisfy conditions 1(abc) must be a justification for the entailment of q therefore is included in T and returned as a positive justification ($\mathcal{J}_{+}$). Since no axioms satisfying conditions 2(abcde) can be found negative justification ($\mathcal{J}_{-}$) is empty.

Case 2: $Q \leftarrow \textbf{not}\, q$

Soundness: In this case, there are two sub-cases: either $\neg q$ is true or not. If it is the former case then the external algorithm JustifyAxiomEntailment is called to compute all justifications, i.e., T, for the entailment of axiom $\neg q$ and for each each $t \in T$ a justification $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle = \langle t, \emptyset \rangle$ is generated. By the definition of axiom entailment justification given in Section 6.1.1, $t \subseteq \mathcal{K}$, $t \models \neg q$, and $\forall t' \subset t, t' \not\models \neg q$. Since negation $\neg$ is strong than negation as failure **not** we have $t \subseteq \mathcal{K}$, $t \models \textbf{not}\, q$, and $\forall t' \subset t, t' \not\models \textbf{not}\, q$. Therefore t satisfies conditions 1(abc) and t is a positive justification ($\mathcal{J}_+$). Also due to the monotonic property of DLs, the entailment of axiom $\neg q$ by $\mathcal{K}$ would always be true no matter what axioms are added to $\mathcal{K}$. That is, there are no non-empty set of axioms satisfy conditions 2(abcde) therefore the negative justification ($\mathcal{J}_-$) is empty. If it is the latter case, a justification $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle = \langle \emptyset, \{q\} \rangle$ is generated. In this case, the truthness of $\textbf{not}\, q$ has nothing to do with the existence of any axioms in $\mathcal{K}$ therefore the positive justification ($\mathcal{J}_+$) is empty. And the truthness of $\textbf{not}\, q$ is due to the absence of $\{q\}$ from $\mathcal{K}$. It is easy to verify $\{q\}$ satisfies conditions 2(abcde) and $\{q\}$ is a negative justification ($\mathcal{J}_-$).
Completeness: For the first sub-case, any set of axioms satisfy conditions 1(abc) must be a justification for the entailment of $\neg q$ therefore is included in T and returned as a positive justification ($\mathcal{J}_+$). Since no axioms satisfying conditions 2(abcde) can be found negative justification ($\mathcal{J}_-$) is empty. For the second sub-case, since no axioms satisfying conditions 1(abc) can be found positive justification ($\mathcal{J}_+$) is empty. And only $\{q\}$ satisfies conditions 2(abcde), therefore there is only one negative justification ($\mathcal{J}_-$), i.e., $\{q\}$.

Case 3: $Q \leftarrow \textbf{not}\,\textbf{not}\, A'$

Soundness & Completeness: In this case, the two **not** operators are removed and the justifications for the truthness of $\textbf{not}\,\textbf{not}\, A'$ are same as the justifications for the truthness of A' which is trivially true.

Case 4: $Q \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$

Soundness: In this case, the justifications $\mathcal{J}_i = \langle \mathcal{J}_{i_+}, \mathcal{J}_{i_-} \rangle$ for each of the n query atoms A_i (i=1,. . .,n) are computed via Alg 5 and combined as justifications for Q via Alg 7: all n justifications' positive parts are combined as $\mathcal{J}_+$ and minimized to $\mathcal{J}_{min}$, for each justification $\mathcal{J}_i$, a $\mathcal{J} = \langle \mathcal{J}_{min}, \mathcal{J}_{i_-} \rangle$ is generated as a justification for

Q. We can see, according to the justification definition, the positive justifications $\mathcal{J}_{i_+}$ (resp. negative justifications $\mathcal{J}_{i_-}$) of the n atoms A_i either satisfy conditions 1(abc) (resp. 2(abcde)) or are empty. Based on this, it is easy to verify $\mathcal{J}_+$ is a positive justification for Q, i.e., satisfying conditions 1(abc). Since $\mathcal{J}_{min}$ is obtained from $\mathcal{J}_+$ via helper function min, $\mathcal{J}_{min}$ is a minimal subset of $\mathcal{J}_+$ that is also a positive justification for Q. Additionally, the negative justification $\mathcal{J}_{i_-}$ of each atom A_i is also a negative justification for Q, i.e., satisfying conditions 2(abcde), which is trivially true. Therefore $\mathcal{J} = \langle \mathcal{J}_{min}, \mathcal{J}_{i_-} \rangle$ is a sound justification.

Completeness: Given a justification $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle$ of Q, according to the justification definition, $\mathcal{J}_+$ (resp. $\mathcal{J}_-$) satisfies conditions 1(abc) (resp. 2(abcde)) or is empty. It is easy to verify that $\mathcal{J}_+$ entails each atom A_i (i=1,. . .,n) and $\mathcal{J}_+$ should be generated from the n positive justifications of atom A_i as Alg 7 does. Similarly it can be verified $\mathcal{J}_-$ is a negative justification of one of the n atoms.

Case 5: $Q \leftarrow A_1 \vee A_2 \vee \ldots \vee A_n$

Soundness: In this case, the justifications $\mathcal{J}_i = \langle \mathcal{J}_{i_+}, \mathcal{J}_{i_-} \rangle$ for each of the n query atoms A_i (i=1,. . .,n) are computed via Alg 5 and combined as justifications for Q via Alg 7: all n justifications' negative parts are combined as $\mathcal{J}_-$ and minimized to $\mathcal{J}_{min}$, for each justification $\mathcal{J}_i$, a $\mathcal{J} = \langle \mathcal{J}_{i_+}, \mathcal{J}_{min} \rangle$ is generated as a justification for Q. The proof of this case is similar to Case 4 except that, instead of proving $\mathcal{J}_+$ thus $\mathcal{J}_{min}$ is a positive justification for Q in Case 4, in this case we need to prove $\mathcal{J}_-$ thus $\mathcal{J}_{min}$ is a negative justification for Q, and instead of proving the negative justification $\mathcal{J}_{i_-}$ of each atom A_i is also a negative justification for Q in Case 4, in this case we need to prove the positive justification $\mathcal{J}_{i_+}$ of each atom A_i is also a positive justification for Q.

Completeness: The proof of this part is also similar to Case 4 except that, given a justification $\mathcal{J} = \langle \mathcal{J}_+, \mathcal{J}_- \rangle$ for Q, instead of proving $\mathcal{J}_+$ can always be generated from the positive justifications of n atoms via Alg 7 in Case 4, in this case we need to prove $\mathcal{J}_-$ can always be generated from the negative justifications of n atoms via Alg 7. Similarly it can be verified $\mathcal{J}_+$ is a positive justification of one of the n atoms.

Case 6: $Q \leftarrow$ **not** $(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$

Soundness: In this case, there are three sub-cases: (1) One of two sub-cases (Q_a,

Q_b as specified in Alg 6) is true. In this sub-case, the justifications of the true query (Q_a or Q_b) are computed and returned as justifications for Q. Given a justification $\mathcal{J}$ of the true query (Q_a or Q_b) it can be verified that $\mathcal{J}$ is also a justification of Q. (2) All atoms A_i are false. In this sub-case, the justifications for the falseness of each atom, i.e., $\textbf{not}\, A_i$, are computed and combined as justifications for Q via 7. The soundness can be proved similarly to Case 5. (3) The conjunction of some atoms from A_i, i.e., Q_p, is true but with additional atoms from A_i, i.e., Q_n, the conjunction becomes false. That is $Q_p \wedge \textbf{not}\, Q_n$ is true. In this case, there are at most $\sum C(n,m)_{m=1,\ldots,n-1} = (2^n - 2)$ combinations of $Q_p \wedge \textbf{not}\, Q_n$ to check. For each true combination, the justifications for this combination are returned as justifications for Q. Given a justification $\mathcal{J}$ for a combination it can be verified that $\mathcal{J}$ is also a justification for Q.

Completeness: Given a query Q, it always fits into one of the three sub-cases, and any justification of Q is always captured and returned by one of the solutions to the three sub-cases.

Case 7: $Q \leftarrow \textbf{not}\,(A_1 \vee A_2 \vee \ldots \vee A_n)$

Soundness & Completeness: Due to the semantics of abbreviation $\vee$, $A_1 \vee A_2 \vee \ldots \vee A_n$ is semantically equivalent to $\textbf{not}\,(\textbf{not}\, A_1 \wedge \textbf{not}\, A_2 \wedge \ldots \wedge \textbf{not}\, A_n)$, therefore $\textbf{not}\,(A_1 \vee A_2 \vee \ldots \vee A_n)$ is equivalent to $\textbf{not}\,\textbf{not}\,(\textbf{not}\, A_1 \wedge \textbf{not}\, A_2 \wedge \ldots \wedge \textbf{not}\, A_n)$, i.e., $\textbf{not}\, A_1 \wedge \textbf{not}\, A_2 \wedge \ldots \wedge \textbf{not}\, A_n$. The justifications for Q are same as justifications for $\textbf{not}\, Q_1 \wedge \textbf{not}\, Q_2 \wedge \ldots \wedge \textbf{not}\, Q_n$.

[1] R. Angles and C. Gutierrez. The Expressive Power of SPARQL. In *ISWC2008*, pages 114–129, 2008.

[2] F. Baader, S. Brand, and C. Lutz. Pushing the EL envelope. In *IJCAI' 2005*, pages 364–369, 2005.

[3] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope Further. In *OWLED' 2008 DC*, 2008.

[4] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *IJCAI91*, pages 452–457, 1991.

[5] F. Baader, R. Kusters, and F. Wolter. Extensions to Description Logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, pages 226–268. Cambridge University Press, 2003.

[6] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, pages 47–100. Cambridge University Press, 2003.

[7] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69(1):5–40, 2001.

[8] D. Beckett and B. McBride. RDF/XML Syntax Specification, 2004.

[9] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1999.

[10] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[11] A. Borgida, D. Calvanese, and M. Rodriguez-Muro. Explanation in DL-Lite. In *DL'2008*, 2008.

[12] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2004.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[14] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective First-Order Query Processing in Description Logics. In *IJCAI'2007*, pages 274–279, 2007.

[15] D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *PODS'98*, pages 149–158, 1998.

[16] L. Ding, J. Tao, and D. L. McGuinness. An Initial Investigation on Evaluating Semantic Web Instance Data. In *WWW' 2008*, pages 1179–1180.

[17] L. Ding, J. Tao, and D. L. McGuinness. OWL Instance Data Evaluation, 2008.

[18] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An Epistemic Operator for Description Logics. *AI*, 100(1–2):225–274, 1998.

[19] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding Epistemic Operators to Concept Languages. In *KR' 92*, pages 342–353, 1992.

[20] F. M. Donini, D. Nardi, and R. Rosati. Autoepistemic Description Logics. In *IJCAI' 97*, pages 136–141, 1997.

[21] F. M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Trans. Comput. Logic*, 3(2):177–225, 2002.

[22] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. *AI*, 172(12-13):1495–1539, 2008.

[23] T. Eiter, C. Lutz, M. Ortiz, and M. Simkus. Query Answering in Description Logics with Transitive Roles. In *IJCAI'2009*, pages 759–764, 2009.

[24] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic SHIQ. In *IJCAI'2007*, pages 399–404, 2007.

[25] B. Glimm and S. Rudolph. Status QIO: Conjunctive Query Entailment Is Decidable. In *KR'2010*, 2010.

[26] P. Hitzler, M. Krotzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language Primer, 2009.

[27] M. Horridge, B. Parsia, and U. Sattler. Laconic and Precise Justifications in OWL. In *ISWC' 2008*, pages 323–338, 2008.

[28] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible SROIQ. In *KR'2006*, pages 57–67. AAAI Press, 2006.

[29] I. Horrocks, B. Motik, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In *ISWC'2006*, pages 501–514, 2006.

[30] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI'00*, pages 399–404, 2000.

[31] A. Kalyanpur, B. Parsia, and B. C. Grau. Beyond Asserted Axioms: Fine-Grain Justifications for OWL-DL Entailments. In *DL'2006*, 2006.

[32] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding All Justifications of OWL DL Entailments. In *ISWC/ASWC'2007*, pages 267–280, 2007.

[33] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, 2004.

[34] R. A. Kowalski. Logic for Data Description. In *Logic and Data Bases*, pages 77–103, 1977.

[35] J. W. Lloyd. *Foundations of logic programming.* 1987.

[36] J. W. Lloyd and R. W. Topor. Making Prolog more Expressive. *J. of Logic Programming*, 1:225–240, 1984.

[37] J. W. Lloyd and R. W. Topor. A Basis for Deductive Database Systems. *J. of Logic Programming*, 2:93–109, 1985.

[38] C. Lutz, D. Toman, and F. Wolter. Conjunctive Query Answering in the Description Logic EL Using a Relational Database System. In *IJCAI'09*, pages 2070–2075, 2009.

[39] D. L. McGuinness and A. T. Borgida. Explaining Subsumption in Description Logics. In *IJCAI'1995*, pages 816–821. Morgan Kaufmann, 1995.

[40] B. Motik. A Faithful Integration of Description Logics with Logic Programming. In *IJCAI'2007*, pages 477–482, 2007.

[41] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles, 2009.

[42] B. Motik, I. Horrocks, and U. Sattler. Bridging the Gap between OWL and Relational Databases. In *WWW'2007*, pages 807–816. ACM Press, 2007.

[43] B. Motik, P. F. Patel-Schneider, and B. C. Grau. OWL 2 Web Ontology Language Direct Semantics, 2009.

[44] B. Motik, P. F. Patel-Schneider, and B. C. Grau. Owl 2 web ontology language direct semantics, 2009.

[45] J.-M. Nicolas and H. Gallaire. Data Base: Theory vs. Interpretation. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 35C–54. Plenum Press, 1978.

[46] J.-M. Nicolas and K. Yazdanian. Integrity Checking in Deductive Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 325–344. Plenum Press, 1978.

[47] J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *ISWC' 2006*, pages 30–43, 2006.

[48] A. Poggi, D. Lembo, D. Calvanese, M. Lenzerini, and R. Rosati. Linking Data to Ontologies. *J. on Data Semantics*, pages 133–173, 2008.

[49] A. Polleres. From SPARQL to rules (and back). In *WWW' 2007*, pages 787–796, 2007.

[50] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf, 2008.

[51] R. Reiter. Towards A Logical Reconstruction of Relational Database Theory. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages (Topics in Information Systems)*, pages 191–233. Springer, 1984.

[52] R. Reiter. On Integrity Constraints. In *TARK'1988*, pages 97–111. Morgan Kaufmann, 1988.

[53] R. Reiter. On Asking What A Database Knows. In J. W. Lloyd, editor, *Computational Logics: Symposium Proceedings*, pages 96C–113. Springer, 1990.

[54] R. Rosati. On the Semantics of Epistemic Description Logics. In *Description Logics*, pages 185–188, 1996.

[55] R. Rosati. Autoepistemic Description Logics. *AI Commun.*, 11(3-4):219–221, 1998.

[56] A. Schaerf. Reasoning with Individuals in Concept Languages. *Data and Knowledge Engineering*, 13:141–176, 1994.

[57] M. Schmidt-Schauß and G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 1(48):1–26, 1991.

[58] M. Schneider. OWL 2 Web Ontology Language RDF-Based Semantics, 2009.

[59] E. Sirin and B. Parsia. Optimizations for answering conjunctive abox queries. In *DL'2006*, 2006.

[60] E. Sirin and J. Tao. Towards Integrity Constraints in OWL. In *OWLED'2009*, 2009.

[61] M. K. Smith, C. Welty, and D. McGuiness. OWL Web Ontology Language Guide, 2004.

[62] J. Tao. Adding Integrity Constraints to the Semantic Web for Instance Data Evaluation. In *ISWC' 2010*, pages 330–337, 2010.

[63] J. Tao, L. Ding, J. Bao, and D. L. McGuinness. Characterizing and Detecting Integrity Issues in OWL Instance Data. In *OWLED' 2008*, 2008.

[64] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity Constraints in OWL. In *AAAI'2010*, pages 1443–1448, 2010.