THEORY AND IMPLEMENTATION OF OBJECT ORIENTED SEMANTIC WEB LANGUAGE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF INFORMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF THE GRADUATE UNIVERSITY FOR ADVANCED STUDIES (SOKENDAI)
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Seiji Koide
2010

MEMBER OF DOCTORAL COMMITTEE

Hideaki Takeda	National Institute of Informatics
Ken Satoh	National Institute of Informatics
Nigel Collier	National Institute of Informatics
Ikki Ohmukai	National Institute of Informatics
Takahira Yamaguchi	Keio University

Approved for the University Committee on Graduate Studies.

Abstract

Resource Description Framework (RDF for short) is an assertional language intended to be used to express propositions using precise formal vocabularies, and its syntax is applicable to the World Wide Webs. RDF Schema (RDFS) is a semantic extension of RDF and it provides a minimum type system to describe web ontologies. OWL Web Ontology Language is a language for defining and instantiating Web ontologies. These three languages for Semantic Webs are intended to be integrated in Semantic Web Layered Architecture, namely OWL was designed to be realized on top of RDF and RDFS. However, this intention is not accomplished and it seems to be coming apart more and more. The objective of this doctoral study is recovering the language integration and provides a unified language system for Semantic Webs.

In this study, firstly, the semantics of RDF, RDFS, and OWL are investigated in common description based on Tarskian denotational semantics, whereas the formal way of describing semantics in W3C Recommendations is different between RDF(S) and OWL. RDF semantics is formalized based on Tarskian denotational model theory and RDFS is extended in the same framework, but OWL semantics is mainly described in the way called Direct Model-Theoretic Semantics, which is appropriate for describing Description Logics and OWL DL. However, due to these two different styles, it has become difficult to understand both languages in the common view, and it has amounted to that OWL has become apart from RDF and RDFS. In this dissertation, an overview of RDF semantics is given in the way described in the RDF documents of W3C Recommendations. Then, OWL semantics is also investigated and formalized based in the same way as RDF, making reference to the OWL specifications in OWL Direct-Model Semantics in the documents of W3C Recommendations. Since our semantic web language system is built on top of Common Lisp Object System, CLOS semantics and its computational model is also discussed. The semantic gap between OWL and object oriented languages are also pointed out.

Secondly, RDF semantics is realized on top of CLOS by straightforward mapping of RDF graph, which is uni-directional labeled graph, to CLOS objects, so that a start node of edge to a CLOS

object, an edge in graph to a slot-name, and an end node of edge to a slot-value. RDFS class/instance relationship is mapped to that in CLOS, and RDFS class/superclass relationship is mapped to that in CLOS, because the semantics of RDFS is analogous to the semantics of CLOS type system. The problems arising from such straightforward mapping for RDF and RDFS are discussed and solved in the realization of our language system for Semantic Webs. Then, all OWL features are implemented on top of RDF(S) by CLOS with preserving RDF(S) semantics. We distinguish substantial sorts and non-substantial sorts in ontology description, and procedural subsumption computation algorithm for OWL Full is developed. The system is named SWCLOS from the acronym of Semantic Web Common Lisp Object System.

Thirdly, the efficiency of SWCLOS implementation was tested by the Lehigh University Benchmark (LUBM) test, and SWCLOS showed the comparable performance to other OWL reasoners, which are reported in the LUBM report. SWCLOS returned correct answers for all LUBM queries, whereas two reasoners out of three returned wrong answers for some queries in the benchmark report. Distinctive benchmark results of SWCLOS were analyzed and improvements of efficiency were achieved by several different engineering methods. The metamodeling capability of SWCLOS was also demonstrated in some examples of SWCLOS metamodeling programming.

Through this study, we obtained deep understanding of semantics on RDF, RDFS, and OWL, because it was necessary to realize the integration of semantic web language, namely, OWL Full, in order to solve the semantic disparity between RDF(S) and OWL DL. For example, the subsumption in class hierarchy is weak in RDFS but it is strong in OWL DL. The semantics of OWL DL class is akin to set theory, but the semantics of RDFS class is based on but different from set theory, rather it is close to frame systems. The semantics of RDF(S) is basically categorized into higher order logic but OWL DL is notably a subset of first order logic. RDFS allows the membership loop and enables metamodeling of ontology but OWL DL cannot accept the membership loop and does not allow metamodeling. Entities in RDF universe stand in the Unique Name Assumption (UNA) for graph nodes but entities in OWL universe does not stand in UNA for objects in ontology. RDF semantics is not developed up to Open World but OWL semantics assumes it for WWWs. These highly conceptual and technical issues must be discussed and settled in order to integrate RDF(S) and OWL. The solution for membership loop, weak/strong subsumption, and non-Unique Name Assumption, Open World Assumption are proposed and implemented in SWCLOS.

In addition to these differences of semantic foundation of languages, what is worse, a misunderstanding on the interpretation of RDF semantics involved the community in the theoretical disorder against the discussion of RDF compatibility of OWL. Excessive materialization of RDF entities in

OWL was coached after the argument over ‘comprehension principle’, and RDF semantics has become the focus of criticism under the pretense that ‘comprehension principle’ allows the paradox to invade upon systems. Such theoretical disorders in Semantic Web community are also discussed in order to rescue OWL Full theory from the theoretic disorder. It deserves to know that OWL 2 specifications of W3C Recommendations retracted the term ‘comprehension principle’ with no account from the documentation of W3C in the end.

As its name implies, SWCLOS is not based on a logic system but based on Common Lisp Object System. It is semantically an amalgamation of CLOS and OWL on top of RDF; nevertheless it still conforms to object-oriented paradigm as programming language. It is the reason why we call it object-oriented semantic web language. The ground of enabling SWCLOS can be summarized as follows. First, the subsumption of CLOS is the same as the subsumption of RDF(S), and the structure of hierarchy and orders of CLOS classes is the same as ones of RDF(S). The dynamic property of CLOS programming and the Meta-Object Protocol of CLOS enabled to tailor the semantics of language within the realms of CLOS language. In fact, it was easy to realize RDF semantics on top of CLOS, because the semantics is almost same except property-centric or object-centric. Then, OWL Full level capability is obtained in OWL by pursuing the compatibility to RDF and preserving it.

[This page intentionally left blank]

Preface

After the first International Semantic Web Conference at 2002 Sardinia, which I attended with the purpose of surveying the feasibility of Semantic Technology for an ICT project started at FY 2002 in Japan, in which I engaged myself, Semantic Webs have been becoming reality more and more year by year. However, it is still far from the goal Tim Berners-Lee intended, and many problems are left to be solved. The most urgent issue on Semantic Webs to be solved is, in my opinion, to establish OWL Full language, which is a unified language of OWL and RDF(S). Whereas the integration of Semantic Web languages of RDF, RDFS, and OWL is the course laid by Tim Berners-Lee in order to establish Semantic Webs, the possibility of OWL Full seems to be rather decreased in W3C along with the publication of the recommendations of OWL 1 at 2004 to the new recommendation of OWL 2 at 2009. It may have been owing to the semantic gaps between RDF and OWL or owing to some historical accidents.

There have been arguments up to now between two camps in Semantic Web community. One side often criticizes the undecidability contained in RDF(S) from the standpoint of first order logic, more precisely Description Logics. The most arguable point was the membership loop of classes, or broadly speaking self-referencing nature of the semantics. It threatens ones like vicious circle. However, the study of reflective systems has been one of the most attractive themes for researchers in Artificial Intelligence over three decades, and then many reflective systems have been proposed and developed mainly in two computer languages, Lisp and Prolog. Common Lisp Object System (CLOS) is the first result in a practical view as a reflective object oriented programming system. The Meta-Object Protocol (MOP) of CLOS facilitates programmers to change the behavior of a CLOS language system within the realms of itself. Needless to say, recursive programming is the most elemental programming way in Lisp. Lisp programmers are familiar with the idea of self-reference and do not have any bewilderment on reflection systems. Thus, it was obvious for lisp programmers who are accustomed to MOP programming that OWL Full was realizable through MOP by implementing the membership loop of rdfs:Class and changing the language semantics to

RDF(S) and OWL.

This study is on the theory of OWL Full language and its implementation on top of CLOS. OWL semantics is re-organized here according to Tarskian denotational model theory in the same way of RDF semantics for the sake of compatibility to RDF semantics, and it is integrated to RDF semantics. OWL on top of RDF(S) is actually realized using MOP of CLOS. Then, the system has been called SWCLOS.

SWCLOS is unique among several plausible realizations of OWL system. It is neither logic-based nor rule-based, rather an object-based system. Every entity of RDF(S) and OWL and user-defined entity in SWCLOS is an object of CLOS. Furthermore, it still conforms to object oriented programming paradigm. It is not a specific application system dedicated for Semantic Webs, rather it is a programming language system for semantic web applications. Programmers can build their owl applications for Semantic Webs using SWCLOS. It is an amalgamation of OWL and an object oriented programming language CLOS.

On the soundness and completeness of the system, SWCLOS may be sound if it does not include bugs, but it is difficult to prove on the completeness of computation. Many entailment and axiomatic rules of RDF(S) and OWL are procedurally implemented in program code as parts of methods in object oriented programming. More precisely, some of them are partly implemented as default reasoning, some of entailment rules are partly implemented as inconsistency checking, but all of them are separately scattered as procedural program in SWCLOS. However, it deserves to know that critiques on undecidability of RDF(S) are often not valid from the viewpoint of engineering. Concrete examples raised as evidence that shows the undecidability are always sick and make no sense. In SWCLOS, the criteria for metamodeling and membership loop are proposed. They assure the soundness of metamodeling from the view of engineering, and it is compatible to CLOS metamodeling facility.

Obviously, RDF and OWL are legitimate successors of Knowledge Representation (KR) systems that were enthusiastically studied in late 1970s and early 1980s as a kind of Artificial Intelligence systems. Even in today, the study of RDF and OWL belongs, as the predecessors did, to the KR discipline as a subfield of Artificial Intelligence. Therefore, the study includes many divergent aspects, i.e., not only symbol logic but also philosophy, cognition, set theory, theory of computation, semantic theory, inference technique, system building engineering, etc. Though Object Oriented Language (OOL) might not be regarded as one of KR languages today, the study of KR systems once helped to form OOLs and also contributed to the formalization of KR languages, from which RDF and OWL inherited the legacy. The inheritance of the KR study still lives in a

large number of today's OOLs as well as in RDF and OWL. Thus, it is plausible to use OOLs in order to represent ontology.

OWL ontology has been showing its potential as normative description of class modeling in object oriented programming. Object Management Group (OMG) has recently issued the adoption of OWL ontology for Model Driven Architecture, and then OWL has become to be unified into UML. However, the systematization of Ontology Driven Architecture with UML is not realized yet.

SWCLOS is the first full-fledged language system as OWL Full processor. The OWL Full metaclass is formalized using by RDF(S) axioms, and the OWL universe is formalized as subset of the RDF universe. The membership loop at the universal class rdfs:Resource is properly implemented under the CLOS metamodeling semantics. The membership loop of rdfs:Class is also properly implemented by introducing a proxy of it in CLOS. Non-Unique Name Assumption in OWL is superimposed onto RDF graph, and the novel algorithm for Unique Name Assumption for atomic objects in the non-UNA condition is invented in order to integrate OWL non-Unique Name Assumption to RDF graph.

This dissertation is structured as follows. Chapter 1 introduces the ground of Semantic Webs and makes clear the motivation of this study. Chapter 2 describes basic semantics of RDF, OWL, and CLOS, mainly with reference of the documents from W3C. The semantic gaps between RDF(S) and OWL are described by Tarskian semantics, which is applied for RDF Semantics, according to the outline of the Direct Model-Theoretic Semantics. This part in Chapter 2 is an original contribution for the denotational semantics of OWL theory. It is also intended to be prepared for Chapter 6 OWL Full theory. Chapter 3 describes an implementation of RDF(S) and OWL on top of CLOS. A benchmark test of SWCLOS is reported in Chapter 4. The criteria for metamodeling using SWCLOS is addressed in Chapter 5, and the example of metamodeling using SWCLOS is also described in Chapter 5. Chapter 6 presents advanced and nicer discussion of OWL Full theory. In this chapter, several kinds of set theories and the comprehension principle in set theories are reviewed from the standpoint of ontology description. Then, it is emphasized that those set theories are not appropriate to ontology description, and it is stated that Russell's Paradox is not applicable for RDF and OWL semantics, rather Russell's Ramified Type Theory is appropriate for class notion in ontology. The problem of combining non-Unique Name Assumption in OWL to RDF graph is also discussed, and a solution is addressed with focusing on graph equality and default reasoning for primitive objects. Chapter 7 pursues Open World Assumption and disjointness of classes in ontologies, then a drawback of the current OWL for ontology description language is pointed out and the direction of solution is suggested. Chapter 8 discusses the related work as overall description of this dissertation.

Chapter 9 makes summary of this dissertation and discusses the future of OWL and SWCLOS. It makes several remarks about the potential of ISO standard Common Logic as ontology description language with the desire of going beyond RDF and OWL.

For readers who matured at Semantic Web theory and practice, it is recommended to pick up Section 2.2 and Chapter 6 through 7. These parts are our original contribution to OWL Full theory. For readers as practitioner of system developers, it is recommended to directly jump into Chapter 3, and pick up Section 2.3 and 2.4. However, the description assumes that readers have some prerequisite knowledge about the reflection and Meta-Object Protocol. Readers who want to know the skill and practice of the reflection, these parts and source codes of SWCLOS are a plenty of the source deserving to dig it.

Acknowledgment

The half of the work of this doctoral study was initially done as a part of the Japanese IT project titled ‘Building a Support System for the Large-Scale Operation System using Information Technology’ under the contract of Galaxy Express Corporation with the Ministry of Education, Culture, Sports, and Technology (MEXT) from FY 2002 through FY 2005. I appreciate two project leaders, Pres. Yukio Kitamura and Pres. Susumu Nagano of Galaxy Express Corporation. After closing the project, Pres. Nagano and Vice Pres. Yoshirou Kondou kindly supported me to continue my work for Semantic Web Technology as Ph.D. student. I thank Emeritus Prof. Setsuo Ohsuga at Tokyo University, who was the chairperson of the Technology Evaluation Committee of the project. I also thank Prof. Riichiro Mizoguchi at Osaka University, who was a co-researcher in the project. Whereas I played the role of supervisor in the project, I worked with competent colleagues, Mr. Norikazu Shimada, Mr. Shohei Misono, and Mr. Masanori Kawamura. All of these experienced colleagues were very helpful for me to achieve the goal of the project. I also appreciate Dr. Ken Kaneiwa, who was not related to the project but the private discussion with him via email on Semantic Web and Wine Ontology was very helpful to understand OWL semantics.

From the viewpoint of engineering, the MEXT project was successfully completed, but I saw that my work was not completed from the viewpoint of academic study. I understood the lack of theoretical generality in my work. The half of this study was carried out as doctoral study at the Graduate University for Advanced Studies (SOKENDAI). The study at SOKENDAI brought the theoretical depth to my work, and then I evolved my work in theory and I achieved the complete realization of object oriented semantic web language in the end, not only in practice but also in theory. The advisor Prof. Hideaki Takeda always stimulated and helped me with his wide-ranged scientific knowledge. I shall be truly grateful to Prof. Takeda. At the meeting, I always enjoyed the discussion with him, and he always gave me a great suggestion when I encountered difficulties, even if he was not aware of it.

I also appreciate CEO of Franz Inc. Jans Aasman and scientist Steve Haflich. As a part of the

ICT project, we worked together twice in a very short term. And it resulted in that I helped Franz to add Semantic Technology into the Franz product line.

Lastly, I would like to ask pardon for disclosing my acknowledgment to my wife Keiko. She is my marvelous partner in my long life and co-fighter of everyday life. Due to my doctoral study at SOKENDAI and the full-time work at Galaxy Express earlier and at IHI Corporation later, I could not help me spending my private time at home to the study. She accepted everything and supported me. This dissertation is dedicated to her.

List of Notations

Notation	*Meaning*
$\langle . \rangle$	Expression for a tuple.
$\{.\}$	Expression for a set.
$\mathcal{I}$	Interpretation that maps a set of URLs to a set of nodes in graph or objects in logics.
$EXT^{\mathcal{I}}(.)$	A mapping from properties in universe of discourse into the powerset of binary pairs of resources in the universe of discourse. It is called an extension of property.
$CEXT^{\mathcal{I}}(.)$	A mapping from classes in universe of discourse into the resources in the universe of discourse. It is called an extension of class.
$\boldsymbol{R}$	Universe of discourse or RDF universe.
$\mathcal{R}$	Russell's class
R(.,.)	Relation
$\boldsymbol{P}$	A set of property in universe of discourse.
$\wp(.)$	Power set
$\mathcal{V}$	vocabulary.
ω	Transfinite ordinal number
$\in$	Expression for membership. $x \in A$ means x is a member of a set A or a class A.
$\sqcap$	Intersection of concepts.
$\sqsubseteq$	Relationship between subconcept and superconcept.
$\top$	Top of concepts.
$\bot$	Bottom of concepts.
$\equiv$	Identical objects.
$\preceq$	Subclass/superclass relationship in CLOS or RDFS.
$\simeq$	Equivalent classes in owl:FunctionalProperty and owl:InverseFunctionalProperty.
$\asymp$	Relationship of complement and disjoint concepts.
$\doteq$	Individual equivalence in OWL.

Glossary of Abbreviations

Abbreviations	*Stands for*
ABox	Assertional Box. See also TBox.
CLCE	Common Logic Controlled English
CGIF	Conceptual Graph Interchange Format
CLIF	Common Logic Interchange Format
CLOS	Common Lisp Object System.
CWA	Closed World Assumption. In CWA, NaF is usually utilized for inference. See also OWA.
DAML	DARPA Agent Markup Language.
DL	Description Logic.
KAON2	KArlsruhe ONtology version 2.
KIF	Knowledge Knowledge Interchange Format
KR	Knowledge Representation.
KRL	Knowledge Representation Language.
LCWA	Local Closed World Assumption.
LUBM	Lehigh University Benchmark.
MDA	Model-Driven Architecture.
MKNF	Minimal knowledge and negation as failure
MOF	Meta-Object Facility.
MOP	Meta-Object Protocol.
NaF	Negation as Failure, the presumption that if *P* is not derived, then *P* is false.
NF	New Foundation
NBG	von Neuman-Bernays-Gödel Set Theory

Abbreviations	*Stands for*
ODA	Ontology Driven Architecture.
ODM	Ontology Definition Metamodel.
OIL	Ontology Inference Layer.
OMG	Object Management Group.
OOL	Object-Oriented Language.
OOP	Object-Oriented Programming.
OOPL	Object-Oriented Programming Language.
OWA	Open World Assumption, the presumption that if neither *P* nor not *P* is derived, then *P* is neither true nor false. See also CWA.
OWL	Web Ontology Language.
OWL 1	OWL version 1.
OWL 2	OWL version 2.
OWL-S	Semantic Markup for Web Services.
OWL DL	OWL by Description Logic.
OWL Full	OWL in full flexibility on syntax and semantics.
OWL Lite	OWL for easy implementation and usage.
PM	Principia Mathematica
RDF	Resource Description Framework.
RDFS	RDF Schema.
RDF(S)	RDF and RDFS.
RTT	Russell's Type Theory
SETF	Software Engineering Task Force.
SPARQL	SPARQL Protocol and RDF Query Language.
SWCLOS	Semantic Web CLOS
TBox	Terminological Box. See also ABox.
UNA	unique name assumption, the presumption that if two names or URLs are different, then those denotation are different.
XCL	XML Common Logic
XML	Extensible Markup Language.
ZF	Zermelo-Fraenkel Set Theory
ZFC	Zermelo-Fraenkel Set Theory with the axiom of Choice

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

> *"I have left open the discussion as to what inference power and algorithms will be useful on the semantic web precisely because it will always be an open question. When a language is sufficiently expressive to be able to express the state of the real world and real problems then there will be no one query engine which will be able to solve real problems.*" (Engines of the future in 'Evolvability', Tim Berners-Lee)

The goal of this doctoral study is a unified language of RDF(S) and OWL for Semantic Webs, namely, the development of an OWL Full language system. Obstacles to achieve this goal must be discovered and removed. In this chapter, in order to make the problems clear for the sake of achieving this goal, we pose an overview of discrepancies between the original goal for Semantic Webs and the current reality. Especially, in addition to the notable discrepancy of metamodeling, we point out the problematic non-Unique Name Assumption and Open World Assumption in OWL. However, the precise discussions of these problems are described after Chapter 6.

1.1 Semantic Web Languages and their Layering

RDF is an acronym of Resource Description Framework and "RDF is an assertional language intended to be used to express propositions using precise formal vocabularies" (RDF Semantics, [25]), and its syntax is applicable to the World Wide Webs (WWWs) with the components such as URI references, literals, and XML schema typed literals.

RDF Schema (RDFS for short) is an semantic extension of RDF and it provides a minimum type system to describe web ontologies on top of RDF. The most remarkable feature of RDFS type

system is the metamodeling capability for ontology description. Namely, using RDFS, ones can treat a class simultaneously as a collection of objects and as an object in its own right.

"The OWL Web Ontology Language is a language for defining and instantiating Web ontologies" (OWL Guide [65]) Due to the design for representing the web content as ontology, it can facilitate greater *machine interpretability* than is supported by XML, RDF, or RDFS, by providing additional vocabulary for concepts. It was intended to be an extension of RDF and RDFS.

This language layering is originally proposed by Tim Berners-Lee. **Fig.1.1** shows the image of such layering called Semantic Layer Cake. In this figure, the layer named 'ontology vocabulary' corresponds to the current OWL.

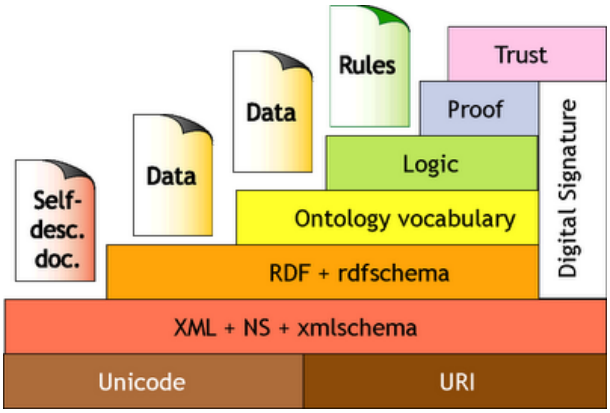

Fig. 1.1: Semantic Web Layer Cake, Original Version.

"The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users." (OWL Guide [65]) OWL Lite is the simplest sublanguage and it supports those users primarily needing a classification hierarchy and simple constraint features. "OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems." (OWL Guide [65]) OWL Lite and DL specifications and its semantics are described in a number of OWL documents [65, 49, 57]. OWL Lite and OWL DL, however, do not support RDF semantics. "OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF" (OWL Guide [65]), but it is not fully developed yet in theory and practice.

As mentioned above, the semantics of OWL DL is not laid on top of RDF semantics. The most remarkable discrepancy, publicly admitted in the community, is the metamodeling capability [19, 27]. For example, in an ontology for the Red Data Book, RDFS and OWL Full can classify a species as a class of creature into the concept of endangered species, but OWL DL cannot perform

such classification, because such metamodeling goes beyond Description Logics and OWL DL. In the followings, `ex:EngangeredSpecies` should be a metaclass due to `ex:Eagle` is a class.

```
<owl:Class rdf:about="&ex;Eagle">
   <rdf:type rdf:resource="&ex;EndangeredSpecies" />
</owl:Class>

<owl:Thing rdf:about="&ex;Harry">
   <rdf:type rdf:resource="&ex;Eagle />
</owl:Thing>
```

The problem of OWL DL that is based on Description Logics is the strict separation between the class and the individual in ontology. Borgida et al. pointed out that one must create a 'meta-individual' [6] in order to work around such a problem in Description Logics. The OWL Working Group in W3C had made efforts to introduce into OWL the metamodeling facility with the terminology of *punning*[1], and it was formally recommended at OWL 2[2]. However, the detail of specification for punning is removed from OWL 2 [27] and no logic based modeling language system appears yet that allows OWL Full metamodeling.

The syntactic freedom of RDF is also strictly limited in OWL DL. RDF and OWL Full can annotate an entity of property using the property itself as follows, but OWL DL cannot perform such annotation, because OWL individuals, classes, and properties are pairwise disjoint [58]. See the following example, which is taken from RDF definition file, 22-rdf-syntax-ns.rdf.

```
<rdf:Property rdf:about="&rdfs;label">
  <rdfs:label>label</rdfs:label>
</rdf:Property>
```

1.2 Intensional Model and Extensional Model

RDF semantics utilizes the notion of set in set theory to formalize its denotational semantics. However, "the use of set-theoretic language here is not supposed to imply that the things in the universe are set-theoretic in nature." (RDF Semantics [25]) On the other hand, the class notion in OWL DL semantics is the same as set in set theory. In the RDF semantics, an extension of a property and an extension of class are formalized using Tarskian denotational model theory. "The use of the explicit extension mapping also makes it possible for two properties to have exactly the same values, or two classes to contain the same instances, and still be distinct entities." (RDF Semantics [25]) Namely,

[1] `http://www.w3.org/2007/OWL/wiki/Punning`
[2] `http://www.w3.org/TR/owl2-new-features/#F12:_Punning`

we can hold two different classes that have the same set as class extension. On the other hand, two classes that have the same set as individuals are regarded as equivalent in OWL DL semantics. This different semantics often causes confusion of the interpretation on rdfs:subClassOf or subsumption. For example, while `vin:RedWine` in Wine Ontology is defined as below.

```
<owl:Class rdf:ID="RedWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#Red" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

The following description does not cause any difference as a result in OWL DL semantics, but it causes the different interpretation in RDF semantics.

```
<owl:Class rdf:ID="RedWine">
  <rdfs:subClassOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#Red" />
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

“This means that RDFS classes can be considered to be rather more than simple sets; they can be thought of as ‘classifications’ or ‘concepts’ which have a robust notion of identity which goes beyond a simple extensional correspondence.” (RDF Semantics [25]) However, this statement is not applicable to OWL DL. Thus, the combination of ‘extensional’ RDF and ‘intensional’ OWL was a basic question to be solved for the realization of OWL Full.

1.3 Non-Unique Name Assumption and Open World Assumption

OWL is equipped with two properties owl:sameAS and owl:differentFrom for individuals. It implies that OWL assumes that an identical name stand for an identical object but different names may or may not stand for the same object in ontology. It is unusual convention for computer languages and predicate calculus. Even if we have two different names for individuals in OWL ontology, say, Bush

and Obama, we cannot distinguish both if we have no other information to determine the sameness or differentness. It is problematic in ontology description in OWL. In order to obtain useful interpretations in OWL, we must describe the sameness/differentness information for individuals.

On the other hand, different URI references in RDF denote different nodes in RDF graph in RDF semantics. However, there were no theory and no practical solutions how to superimpose the non-Unique Name Assumption (non-UNA) in OWL onto the RDF graph. It was also a question to be solved for the realization of OWL Full.

In addition to the denotational ambiguity for individuals with respect to the non-UNA, we consider Semantic Webs open in the sense that no one can exhaustively scrape all of the World Wide Webs. It also amounts to a problematic situation. For example, although owl:someValuesFrom restriction requires that the property value exists such that satisfies the restriction on the property for individuals of a class restricted by owl:someValuesFrom, we cannot perform this requirement if we adopt and interpret Open World Assumption (OWA) rigorously, because someone might have defined it somewhere in the WWWs. If so, we cannot conclude the unsatisfiability for owl:someValuesFrom restriction. In fact, Description Logics, which assumes OWA, it is not thorough enough from the viewpoint of semantics in Semantic Webs, and most reasoners for OWL DL seems to be loose of owl:someValuesFrom on OWA[3].

The class notion in OWL DL is the same as set in set theory. Then, the equivalence and disjointness on classes are rather simple in OWL semantics, in case that we can check ABox. However, in case that we have an empty or poor ABox, it is difficult to conclude useful results from TBox, if there is no explicit description with owl:disjointWith or owl:equivalentWith. Thus, OWL ontology of TBox requires to write down that Artifact is disjoint with LivingThing, Aminal is disjoint with Plant, Person is disjoint with Horse, Monkey is disjoint with Ape, Bacteria is disjoint with Virus, etc. This amounts to huge number of lines about disjointness on classes in ontology description.[4]

In the RDF Semantics document [25], there is no description on class disjointness. Simply, if two classes are not related in rdfs:subClassOf and also does not share any subclass, then both can be captured disjoint. However, it does not negate the possibility of entering superclass/subclass relationship or sharing a subclass. Obviously, the semantics in RDFS and OWL on class disjointness are different. So, how to combine both of disjointness in RDFS and OWL was a question to be solved.

[3]This question is discussed in Chapter 8.

[4]Actually, 58% is for class disjointness in lines of pizza.owl for only 23 pizza and 29 pizza toppings. The number of lines for disjointness will explode with the number of classes.

1.4 Perspective of Semantic Web from Object Oriented Programming

1.4.1 UML and Reflection

The work of ontology building is similar to the work of class design in Object-Oriented Programming (OOP). Therefore, it is natural to regard the domain modeling of OOP as *object-centered modeling* [6] in ontology development. Aiming the integration of OWL with OOP, which we expect to produce valid, sound, and reusable OOP software programs on the solid base of OWL, the Software Engineering Task Force (SETF)[5] in W3C Semantic Web Best Practices and Deployment Working Group had been started to promote synergies between the semantic web technology and domains associated with software engineering [35]. Sharing the goal and taking over the work, Object Management Group (OMG) recently standardized Ontology Definition Metamodel (ODM)[1] of the method of software development based on RDF and OWL as the foundation of Model-Driven Architecture (MDA) for Meta-Object Facility (MOF)[6].

The main objective of such activities is to establish the ontology driven software engineering, in which ones expect benefits of unambiguous domain models, consistency checking facilities, validated model sharing, and automatic code generation in software development. However, the realization of the Ontology Driven Architecture (ODA) by SETF [73] requires to reconstruct the framework of the object oriented modeling of software engineering based on Semantic Web, in particular, OWL.

There were two ways to enable OOP upon OWL. One was to establish the class design by UML, which is independent of specific languages, on the framework of OWL [36, 11, 14]. In this case, the metamodeling structure of RDF(S) and OWL must be mapped onto the four layered architecture of the OMG's MOF [4]. "RDFS differs from many conventional ontology frameworks such as UML which assume a more structured hierarchy of individuals, sets of individuals, etc., or which draw a sharp distinction between data and meta-data. However, while RDFS does not assume the existence of such structure, it does not prohibit it." (OWL Semantics [25]) The other way was to enable OWL metamodeling by a specific reflective computer language that allows ones meta-programming within the language [39, 38, 40, 41, 42] instead of separated metamodeling layers with distinct language systems like MOF. With respect to static Object-Oriented Programming Languages (OOPLs) such as C# and Java, it is difficult to change the semantics of languages to meet OWL and then inevitably we cannot but choose the former approach. However, it was feasible that a dynamic and reflective

[5] http://www.w3.org/2001/sw/BestPractices/SE/

[6] http://www.omg.org/mof/

language like Common Lisp Object System (CLOS) [69, 56], in which the CLOS native semantics may be changed to OWL semantics using Meta-Object Protocol (MOP) [31], enables to realize OWL within CLOS.

The OWL Full language was designed to inherit ontology metamodeling characteristics of RDFS. The OWL Guide [65] states that the choice between OWL DL and OWL Full mainly depends on the extent to which users require the metamodeling facilities of RDFS, i.e., defining a class of classes.[7] However, in software engineering, the decision whether as a class or as an instance ones capture an entity depends upon the characteristics of the application domain and modeler's intention. For example, a wine product such as Elyse Zinfandel may be an instance for wine expert systems, but it should be a class in logistics software of wine wholesalers. Thus, an ontology modeling language must allow ones to encode metamodeling ontology due to the requirement of ontology reusability.

CLOS is a reflective OOPL that allows meta-programming in OOP by MOP. Therefore, we expected that the integration of OWL with CLOS that provides metamodeling facilities produced OWL Full capability with OOP metamodeling facilities. In CLOS, a class is not only an object schema to define instances but also an object called *metaobject*. CLOS programmers can encode the metamodeling software with CLOS. Therefore, it was plausible that the performance of OWL Full was obtained by using CLOS meta-programming facilities with SWCLOS.

1.4.2 Type Theory and Metamodeling Criteria

The class hierarchy of CLOS is integrated to Common Lisp type system. The class structure for inheritance mechanism in CLOS is very similar to the notion of RDFS subsumption. Thus, mapping from RDFS classes to CLOS classes is adopted in SWCLOS. Due to this straightforward mapping, the subsumption computation of RDFS is automatically performed by the machinery of CLOS class inheritance. The semantics of RDF types is also implemented as tailored type system of CLOS. Furthermore, the relationship between rdfs:Resource and rdfs:Class is analogous to cl:standard-object and cl:standard-class in CLOS. The semantics contained in this relationship is the same between CLOS and RDFS. As a result, the realization of RDF(S) on top of CLOS was very easy. On the other hand, such isomorphic mapping has amounted to make the metamodeling in RDF(S) enlightened on CLOS metamodeling semantics and capability. Thus, the metamodeling criteria in SWCLOS has been elaborated from the metamodeling semantics of CLOS.

[7] http://www.w3.org/TR/owl-guide#OwlVarieties

1.5 A Realistic Solution with Moderate Semantic Web Conditions

It seems that OWL has successfully established itself as a *de facto* standard of ontology description language not only in the Semantic Web community but also in diverse disciplines and engineering fields, e.g., ontology, linguistics, modeling in software engineering, enterprise business patterns, etc. We developed SWCLOS [41], and attempted to apply it in several applications. Then, we saw how such language that is firmly underpinned by formal logic and denotational semantics is useful to software engineering so as to assure formal descriptions of system specification of applications.

However, in the process of developing SWCLOS we encountered a few subtle and basic problems of semantic distinction between RDF and OWL. We found that full setting of non-UNA and OWA for Semantic Webs amount to either very few viable interpretations with less common knowledge otherwise excessive need of common knowledge for models on class disjointness and individual differentiation in several Semantic Web applications. Hence, we refactored SWCLOS with introducing new moderate settings based on *context dependent role and disjointness of substance classes*, *auto-epistemic local closed world assumption*, *ternary truth values* that allow unknown value, and *UNA for atomic objects in non-UNA environment*. Such experience of developing SWCLOS and the subsequent applications brought us to deeper understanding on the theory and relations of RDF(S)/OWL, logics, and Object-Oriented semantics.

Along with the diffusion of OWL systems and the progress of OWL deployment, it is true that many people have been noticed the limitation of OWL DL specification, and they are becoming to suffer from drawbacks of OWL DL in building ontology. The goal of this doctoral study is to develop an OWL Full language system and recover the original intention for Semantic Webs. We believe this goal is basically accomplished in SWCLOS, a CLOS-based object oriented programming language for Semantic Webs.

1.6 Guide to Readers

It is convenient to suppose typical reader's types in order to make a guidance. For readers who matured at Semantic Web theory and practice, it is recommended to pick up Section 2.2 and Chapter 6 through 7. These parts are our original contribution to OWL Full theory. If someone who is interested in set theories in ontology, which are described in Chapter 6, Appendix A and B will be a good guide for progressing to this domain.

For readers as practitioner of system developers, it is recommended to directly jump into Chapter

3, and pick up Section 2.3 and 2.4. However, the description assumes that readers have some prerequisite knowledge about reflection and Meta-Object Protocol. Readers who are unfamiliar with these notions might feel difficulties. However, readers who want to know the skill and practice of reflection, these parts and source codes of SWCLOS are a plenty source of deserving to dig it.

Unfortunately, for readers as users who are seeking convenient tools for Semantic Webs, SWCLOS is not the case. It is a programming language for Semantic Webs. Users are required to encode their own program that they want to realize their application. This dissertation is also not the case. However, for all who are widely interested in Semantic Webs, we believe this dissertation gives very unique views in Semantic Web languages.

[This page intentionally left blank]

Chapter 2

Semantics of RDF, OWL, and CLOS

> *"Every scientific theory is a system of sentences which are accepted as true and which may be called [. . .] asserted statements or, for short, simply statements."* (Alfred Tarski, [71], p.3)

In this chapter, the semantics of RDF, RDFS, and OWL are studied based on Tarskian denotational semantics. Firstly, an overview of RDF semantics in Tarskian denotational model theory are given in the way described at the RDF documents of W3C Recommendation. Secondly, in order to develop OWL Full theory, OWL semantics is also studied based on Tarskian denotational semantics with the reference of the OWL specifications in the OWL Direct-Model Semantics. Thirdly, CLOS semantics and its computational model are investigated, and the semantic gap between OWL and object oriented languages are pointed out.

2.1 RDF Semantics

2.1.1 Denotational Semantics in RDF

The foundation of formal language is laid by Alfred Tarski [71] along with the concept of denotational semantics. Drew McDermott unreservedly stressed the important role of the denotational semantics in language specification on his paper titled "Tarskian Semantics, or No Notation Without Denotation!" [48] In his paper, he described the basic feature of the denotational semantics as "The method is called as 'denotational' because it specified the meanings of a notation in terms of what its expressions denote." In a typical predicate calculus, we have no problem to represent knowledge

so long as we treat only number theory and mathematical concepts. However, when we use symbols such as *horse*, *car*, *city*, and so forth, in sentences of predicate calculus, we are required to make the meanings of such symbols clear in virtue of predicate calculus. Undoubtedly, the effort to represent general knowledge by predicate calculus involved the problem that exceeded the predicate calculus for mathematical propositions started by Gottlob Frege. In Tarskian Semantics, "we assign to primitive symbols denotations which consist of objects, functions, or predicates. Then the meaning of more complex expressions are defined by rules which define their meanings in terms of the meanings of their parts." (McDermott [48])

In Tarskian Semantics, a symbol (as atomic term) in statements is usually captured as it denotes a factual or hypothesized thing in the world. It is the same usage in everyday languages. For example, in the following sentence, *New York* denotes a city in U.S. named "New York".

New York is a large city.

However, a symbol itself in statements must be distinguished from a thing denoted by the symbol in order to deduce the truth value of asserted statements using rules for terms of symbol in statements. Tarski [71] described the distinction through the usage of quotation for a term as shown below.

**well consists of four letters.*
**Mary is a proper name.*
"well" consists of four letters.
"Mary" is a proper name.

While the first two sentences do not convey the truth value, the last two sentences allow ones to interpret them and bear the truth value. Here, quoted *"well"* and quoted *"Mary"* do not denote anything as object in the world, rather they represent the symbols themselves. In the formal semantic theory of knowledge representation languages, the relations between symbols and their denotations must be interpreted according to rules in a given formal way, as well as relations among denotations in the world. RDF semantics [25] is also specified based on such *Tarskian* Semantics.

2.1.2 Model Theory and Interpretation

RDF semantics uses a basic technique called *model theory* for specifying the semantics of language. Model theory is a formal semantic theory "that the language refers to a world", in which "the minimal conditions that a world must satisfy" is specified "in order to assign an appropriate meaning" of symbols (RDF Semantics [25]). Such minimal conditions are called *semantic conditions*. Generally

speaking, for a set of given semantic conditions, there are a number of concrete possible worlds that satisfy all semantic conditions and provide the truth value to statements. While the mapping from symbols to their denotations is called *interpretation* mapping, a particular realization of the world in model theory is also called *interpretation*. In other words, the *interpretation* mapping allows us to interpret statements and let them turn out a set of *interpretations* of the world in model theory. Thus, *interpretation* means a minimal formal description of a world which is just sufficient to establish the truth value of any expression in logic.

2.1.3 Resources and Properties in RDF Universe

RDF semantics in the RDF document [25] of W3C specifies several precise mappings or interpretations to *universe of discourse*[1] from the description of which the syntax is specified in the document of RDF syntax [34]. In RDF model theory, RDF does model the world as labeled directed graph. A graph is syntactically expressed as a set of triples. A triple $\langle s, p, o \rangle$ consists of *subject*, *predicate*, and *object*. Herein a subject *s*, a predicate *p*, and an object *o* corresponds to a start node, an arc, and an end node of directed graph, respectively. A subject is expressed by either a *URI reference* or a blank *nodeID*. A URI reference can be replaced by the corresponding *QName*, if the URI has a namespace. A blank node has no URI reference and may be designated by a blank nodeID instead of a URI reference. A predicate is expressed by a URI reference. An object is expressed by either a URI reference, a blank nodeID, or a *literal*. A literal is a *plane literal* (a quoted string with/without optional language tag), or an XML datatype expression called *typed literal*. A URI reference that is assigned to an arc of labeled graph is called *property*. Every URI reference and every literal in triples denote a thing in the universe of discourse, and then the denotation is called *resource* in the universe, but "*resource* is treated here as synonymous with 'entity', i.e. as a generic term for anything in the universe of discourse." (RDF Semantics [25])

For a given set of triples, a set of rules and a set of consistent interpretation mappings from a vocabulary to the universe of discourse determine the truth value of each triple and the whole set of triples, namely, the RDF graph represented. An RDF graph may include blank nodes. An RDF graph that does not include blank nodes is called a *ground graph*. In the document of RDF semantics [25], following interpretation and conditions are firstly addressed as RDF simple interpretation

[1] Tarski explained the concept of *universe of discourse* in his book [71] for a particular mathematical theory. If we rephrase it by substituting the mathematical theory with RDF theory, it is said that "Instead of using the general logical concept of individual within [RDF theory], it is sometimes more convenient to specify exactly what is considered an individual thing within the framework of this theory; the class of all those things will then be denoted again by $[\boldsymbol{R}^I]$ and will be called the universe of discourse of the theory." The advanced discussion is held at Chapter 6. Note that the universe of discourse is specified in set theory as $\boldsymbol{R}^I \equiv \{x \in a \mid x = x\}$. See also Appendix A.

and semantic conditions for ground graphs.

Hereafter an interpretation mapping is expressed by mapping $\mathcal{I}$. Thus, a resource that is denoted by a URI reference x in interpretation $\mathcal{I}$ is expressed as $\mathcal{I}(x)$ or $x^{\mathcal{I}}$.[2] The signature of interpretation $\mathcal{I}$ in RDF is a tuple of $\langle \boldsymbol{R}^{\mathcal{I}}, \boldsymbol{P}^{\mathcal{I}}, EXT^{\mathcal{I}}, S^{\mathcal{I}}, L^{\mathcal{I}}, LV \rangle$. Each element in this tuple is described as follows.

RDF simple interpretation

RDF simple interpretation of vocabulary $\mathcal{V}$ is,

1. A non-empty set $\boldsymbol{R}^{\mathcal{I}}$ of entities, called the domain or universe of $\mathcal{I}$.
2. A set $\boldsymbol{P}^{\mathcal{I}}$, called the set of properties of $\mathcal{I}$.
3. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{P}^{\mathcal{I}}$ into the powerset of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$, i.e., a set of sets of pairs $\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle$ with $x^{\mathcal{I}}$ and $y^{\mathcal{I}}$ in $\boldsymbol{R}^{\mathcal{I}}$.

$$\forall p^{\mathcal{I}} \exists x^{\mathcal{I}} \exists y^{\mathcal{I}} [\ p^{\mathcal{I}} \in \boldsymbol{P}^{\mathcal{I}} \wedge x^{\mathcal{I}} \in \boldsymbol{R}^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \boldsymbol{R}^{\mathcal{I}} \quad \Leftrightarrow \quad \{\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in A \mid EXT^{\mathcal{I}}(p^{\mathcal{I}}) = \wp(A)\}\ ]$$

4. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}$ into $\boldsymbol{R}^{\mathcal{I}} \cup \boldsymbol{P}^{\mathcal{I}}$.
5. A mapping $L^{\mathcal{I}}$ from typed literals in $\mathcal{V}$ into $\boldsymbol{R}^{\mathcal{I}}$.
6. A distinguished subset LV of $\boldsymbol{R}^{\mathcal{I}}$, called the set of literal values, which contains all the plain literals in $\mathcal{V}$.

Here, $\boldsymbol{R}^{\mathcal{I}}$, a set of all entities in discussion, is called the universe of discourse. $EXT^{\mathcal{I}}(p^{\mathcal{I}})$ is called *property extension* of $p^{\mathcal{I}}$. A property can make a set of the binary relation on the property between entities in the universe of discourse.

Semantic conditions for ground graph

A ground RDF graph in $\mathcal{I}$ denotes a truth value and it is computed by recursively applying the following conditions for ground triples.

[2] Due to the explicit discrimination between a URI reference and its denotation in this dissertation, such an expression as variable "x" expresses a URI reference, and "$x^{\mathcal{I}}$" represents a resource in the universe of discourse: a URI reference or QName like "rdf:Property" for "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property" stands for the URI reference itself, and the denotation through an interpretation $\mathcal{I}$ is represented by the expression like $rdf : Property^{\mathcal{I}}$ or $ex : New\_York^{\mathcal{I}}$.

{ "*aaa*" $\in \mathcal{V}$ | "*aaa*" is a plane literal without language tag} $\Rightarrow \mathcal{I}$("*aaa*") = *aaa*
{ "*aaa*"@*ttt* $\in \mathcal{V}$ | "*aaa*"@*ttt* is a plane literal with language tag } $\Rightarrow \mathcal{I}$("*aaa*"@*ttt*) = $\langle aaa, ttt \rangle$
{ $e \in \mathcal{V}$ | e is a typed literal } $\Rightarrow \mathcal{I}(e) = L^{\mathcal{I}}(e)$
{ $e \in \mathcal{V}$ | e is a URI reference } $\Rightarrow \mathcal{I}(e) = S^{\mathcal{I}}(e)$
{ $E = \langle s, p, o \rangle$ | $s \in \mathcal{V}$, $p \in \mathcal{V}$, $o \in \mathcal{V}$ }
$\Rightarrow \mathcal{I}(E)$ = true, if $p^{\mathcal{I}} \in \boldsymbol{P}^{\mathcal{I}} \wedge \langle s^{\mathcal{I}}, o^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(p^{\mathcal{I}})$, otherwise $\mathcal{I}(E)$ = false
{ $\boldsymbol{E}$ | $\boldsymbol{E}$ is a ground RDF graph }
$\Rightarrow \mathcal{I}(\boldsymbol{E})$ = false, if $\exists E'^{\mathcal{I}}$ { $E'^{\mathcal{I}}$ = false | $E'^{\mathcal{I}} \in \boldsymbol{E}^{\mathcal{I}}$ }, otherwise $\mathcal{I}(\boldsymbol{E})$ = true.

The last condition means the truth value of an RDF graph is a conjunction of truth values of triples of the RDF graph. The condition before the last one simply claims that every triple within vocabulary $\mathcal{V}$ may make a property extension on a property in $\boldsymbol{P}^{\mathcal{I}}$.

Semantic conditions for blank nodes

Supposing $\mathcal{A}$ is an interpretation mapping to $\boldsymbol{R}^{\mathcal{I}}$ of $\mathcal{I}$ from blank nodes that appear in triples for an RDF graph $\boldsymbol{E}$, additional semantic conditions for unground graph is described as follows.

{ e | e is a blank node } $\wedge \mathcal{A}(e) \Rightarrow [\mathcal{I} + \mathcal{A}](e) = \mathcal{A}(e)$
{ $\boldsymbol{E}$ | $\boldsymbol{E}$ is an RDF graph }
$\Rightarrow \mathcal{I}(\boldsymbol{E})$ = true, if $\exists \mathcal{A}'$ { $[\mathcal{I} + \mathcal{A}'](E)$ | $E^{\mathcal{I}} \in \boldsymbol{E}^{\mathcal{I}}$ } = true, otherwise $\mathcal{I}(\boldsymbol{E})$ = false

The first condition claims that the interpretation mapping $\mathcal{A}$ for a blank node does not change the interpretation for ground graph $\mathcal{I}$. "It simply extends the rules for defining denotations under an interpretation, so that the same interpretation that provides a truth-value for ground graphs also assigns truth-values to graphs with blank nodes, even though it provides no denotation for the blank nodes themselves." (RDF Semantics [25]) The second condition simply requires that such an interpretation exists consistently for a subset of triples that include blank nodes. Thus, the extended interpretation $[\mathcal{I} + \mathcal{A}]$ supports the instance lemma, that is, a graph is entailed by any of its instances (note that an instance of graph G is obtained from a graph G by replacing some or all of the blank nodes in G with non-blank nodes).

The role of blank node is important to make a scope of variable into ontology and to make individuals that hold the same properties and values. For example, the statement of triplets of cat

that are born by an identical parent cat is described as follows.[3]

```
<ex:aParentCat> rdf:type <ex:Cat> .
<ex:aParentCat> <ex:hasChild> _:b001 .
_:b001 rdf:type rdf:Bag .
_:b001 rdf:_1 _:c001 .
_:b001 rdf:_2 _:c002 .
_:b001 rdf:_3 _:c003 .
_:c001 rdf:type <ex:Cat> .
_:c001 rdf:type <ex:Child> .
_:c001 <ex:look> <ex:Cute> .
_:c002 rdf:type <ex:Cat> .
_:c002 rdf:type <ex:Child> .
_:c002 <ex:look> <ex:Cute> .
_:c003 rdf:type <ex:Cat> .
_:c003 rdf:type <ex:Child> .
_:c003 <ex:look> <ex:Cute> .
```

In this case, three blank nodes are denoted by three blank nodeID from `_:c001`, `_:c002`, and `_:c003`, while every blank nodes has the same structure as subgraph.[4]

For another example on blank nodes, suppose that we have everyday temperatures and summarize them later, it is convenient to wrap up everyday temperature using blank nodes in order to make a summary, otherwise we will be involved in troublesome situation such that we must totally reorganize ontologies.
Supposing in some day we have,

```
_:t010 rdf:type <ex:temperature> .
_:t010 <ex:date> "2011-04-01"^^xsd:date .
_:t010 <ex:value> _:t011 .
_:t011 rdf:value "22"^^xsd:integer .
_:t011 <ex:unit> <ex:Centigrade> .
```

and in another day we have similarly,

```
_:t010 rdf:type <ex:temperature> .
_:t010 <ex:date> "2011-04-02"^^xsd:date .
_:t010 <ex:value> _:t011 .
_:t011 rdf:value "23"^^xsd:integer .
_:t011 <ex:unit> <ex:Centigrade> .
```

[3] This syntax is relaxed N-Triples, which is used in [25].

[4] This simple example, in reality, contains the counting problem in ontology. It is clear that the counting problem is beyond the scope of Description Logics and OWL DL, although they provide a device for *counting quantifier*.

Then, we can merge them easily in the way suggested in the document of RDF Semantics [25],

```
_:t010 rdf:type <ex:temperature> .
_:t010 <ex:date> "2011-04-01"^^xsd:date .
_:t010 <ex:value> _:t011 .
_:t011 rdf:value "22"^^xsd:integer .
_:t011 <ex:unit> <ex:Centigrade> .
_:t020 rdf:type <ex:temperature> .
_:t020 <ex:date> "2011-04-02"^^xsd:date .
_:t020 <ex:value> _:t021 .
_:t021 rdf:value "23"^^xsd:integer .
_:t021 <ex:unit> <ex:Centigrade> .
```

In this case, each of blank nodes `_:t010` and `_:t020` makes a boundary of scoping date and temperature, so that the consistency of date and temperature is preserved in the simple way of merging graphs.

Rdf-interpretation for rdf-vocabulary

In addition to the RDF simple interpretation and the extended interpretation for blank node described above, rdf-interpretation provides basic meanings to built-in rdf-vocabulary, which contains rdf:type, rdf:Property, rdf:XMLLiteral, etc., all of them have the prefix 'rdf'. Especially among them, the set of properties and the notion of property that is previously introduced by the RDF simple interpretation is redefined in RDF semantic conditions using two terminologies in rdf-vocabulary, rdf:Property and rdf:type.

Definition of property (a part of RDF semantic conditions)

If an entity is a member of a set of properties of $\mathcal{I}$, then the entity itself makes a pair with $rdf : Property^{\mathcal{I}}$, and then the pair is a member of property extension of $rdf : type^{\mathcal{I}}$, and vice versa:

$$x^{\mathcal{I}} \in \boldsymbol{P}^{\mathcal{I}} \quad \Leftrightarrow \quad \langle x^{\mathcal{I}}, rdf : Property^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf : type^{\mathcal{I}}) \tag{2.1}$$

This definition also entails that any $x^{\mathcal{I}}$ in $\boldsymbol{P}^{\mathcal{I}}$ and $rdf : Property^{\mathcal{I}}$ belong to $\boldsymbol{R}^{\mathcal{I}}$ along with the RDF simple interpretation.

Proof. From the RDF simple interpretation 3, $\langle x^{\mathcal{I}}, rdf : Property^{\mathcal{I}} \rangle$ is a member of a power set of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$. Then, $\langle x^{\mathcal{I}}, rdf : Property^{\mathcal{I}} \rangle$ is a member of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$. Then, $x^{\mathcal{I}}$ and $rdf : Property^{\mathcal{I}}$ is a member of $\boldsymbol{R}^{\mathcal{I}}$.

Therefore, it is possible to state that p (as predicate) of p (as property resource) is o. For example, it can be stated such as $\langle rdfs:comment^{\mathcal{I}}$, "A description of the subject resource."$\rangle \in EXT^{\mathcal{I}}(rdfs:comment^{\mathcal{I}})$, and it could be expressed in logical expression of predicate calculus as follows,

$$rdfs{:}comment(rdfs{:}comment, \text{"A description of the subject resource."}).$$

Obviously, such self-referential statements are problematic in first order logic and Description Logics. Note that $\boldsymbol{P}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$ in RDF, but $\boldsymbol{P}^{\mathcal{I}}$ and $\boldsymbol{R}^{\mathcal{I}}$ are disjoint in OWL DL.

2.1.4 Semantics of Class in RDF Schema

RDF Schema (RDFS) is a semantic extension of RDF so as to provide mechanisms for describing ontology using the notion of *class* and *instance*. In RDFS, "Resources may be divided into groups called classes. The members of a class are known as instances of the class. [...] The rdf:type property may be used to state that a resource is an instance of a class." (RDF Schema [7]) A set of instances of a particular class is called a *class extension* of the class in RDF semantics.

There are two serious problems on the class notion and the class/instance relationship. Firstly, although the same terminology of 'class' is used in several different disciplines, all the semantics are different each other among set theory, Semantic Web, and object oriented languages. In RDFS, a class is distinguished from a set of instances, while sets of resources are associated with classes in the universe of discourse. "Two classes may have the same set of instances but be different classes." (RDF Schema [7]) So, the class in RDFS enables ones to capture different aspects of an identical set of objects in the world and can categorize them in ontology. However, in OWL, the class is identical to a set. Meanwhile, a class (called proper class) in set theory is not a set. Thus, the concept of class is a confounding and disputable issue among those disciplines[5] Moreover, non-Unique Name Assumption for individuals in OWL gives impetus to the confusion on the notion of the sameness of individuals[6].

Secondly, in RDFS, "a class may be a member of its own class extension and may be an instance of itself." (RDF Schema [7]), while Description Logics and OWL DL cannot accept such *membership loop* of class and instance, because it goes beyond the scope of first order logic and Description Logics.[7] In this subsection on RDFS, the concept of class and instance in RDFS is formally introduced, precisely on concepts for terms, *membership*, *membership loop*, and *subsumption*.

[5] See also Chapter 6.

[6] Some part of this issue is discussed at Section 6.4.

[7] What is worse, there is a misunderstanding in a part of Semantic Web community about 'comprehension principle' and 'Russell's paradox'. The discussion is held at Chapter 6.

Class and instance relation in RDFS semantic conditions

If an entity is a member of class extension of another entity, then a pair of both becomes a member of property extension of $rdf:type^{\mathcal{I}}$, and vice versa.

$$x^{\mathcal{I}} \in CEXT^{\mathcal{I}}(y^{\mathcal{I}}) \quad \Leftrightarrow \quad \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf:type^{\mathcal{I}}) \tag{2.2}$$

Here, $CEXT^{\mathcal{I}}$ is a mapping of $\mathcal{I}$ from $\boldsymbol{C}^{\mathcal{I}}$, a set of all classes in the universe, to a set of subsets of entities in the universe $\boldsymbol{R}^{\mathcal{I}}$. $CEXT^{\mathcal{I}}(y^{\mathcal{I}})$ represents the class extension of $y^{\mathcal{I}}$ called *instances* of $y^{\mathcal{I}}$. We call $y^{\mathcal{I}}$ a *class* in this context.

After obtaining such concept of class and instance, it is obvious that every property in $\boldsymbol{P}^{\mathcal{I}}$ turns out an instance of $rdf:Property^{\mathcal{I}}$.

$$\boldsymbol{P}^{\mathcal{I}} \quad = \quad CEXT^{\mathcal{I}}(rdf:Property^{\mathcal{I}}). \tag{2.3}$$

Proof. Substituting $y^{\mathcal{I}}$ with $rdf:Property^{\mathcal{I}}$ in (2.2) yields the right-hand side of (2.1). Therefore, for every x that satisfies (2.1) and (2.2), (2.3) holds.

Note that $rdf:type^{\mathcal{I}}$ can be also redefined as an instance of $rdf:Property^{\mathcal{I}}$.

$$\langle rdf:type^{\mathcal{I}}, rdf:Property^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf:type^{\mathcal{I}}) \tag{2.4}$$

This definition goes again beyond the scope of first order logic and Description Logics. The first occurrence of $rdf:type^{\mathcal{I}}$ in (2.4) is a node in RDF graph, and the second occurrence stands for an edge named $rdf:type$.

Universal class

Furthermore, using new terminology in rdfs-vocabulary we can name several basic classes in the universe of discourse. Firstly, the universe $\boldsymbol{R}^{\mathcal{I}}$ itself is named as the class extension of $rdfs:Resource^{\mathcal{I}}$.

$$\boldsymbol{R}^{\mathcal{I}} \quad = \quad CEXT^{\mathcal{I}}(rdfs:Resource^{\mathcal{I}}) \tag{2.5}$$

This entity rdfs:Resource is called the universal class, to which every entities in the universe is classified and of which every class in the universe is a subclass.[8]

The combination of RDF semantic condition (2.2) and RDF simple interpretation entails that not only every member of $rdfs:Resource^{\mathcal{I}}$ is an entity as a resource but also $rdfs:Resource^{\mathcal{I}}$ itself is in the universe of discourse.

$$rdfs:Resource^{\mathcal{I}} \quad \in \quad \boldsymbol{R}^{\mathcal{I}} = CEXT^{\mathcal{I}}(rdfs:Resource^{\mathcal{I}}) \tag{2.6}$$

Proof. By substituting $y^{\mathcal{I}}$ in (2.2) with $rdfs : Resource^{\mathcal{I}}$, we obtain $\langle x^{\mathcal{I}},\ rdfs : Resource^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf:type^{\mathcal{I}})$. Then, $rdfs:Resource^{\mathcal{I}}$ belong to the universe $\boldsymbol{R}^{\mathcal{I}}$, because $\langle x^{\mathcal{I}},\ rdfs:Resource^{\mathcal{I}} \rangle$ are members of a powerset of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$ by RDF simple interpretation 3, then they are also members of a set of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$.

Note that (2.6) represents the membership loop of rdfs:Resource, and it goes beyond the scope of first order predicate calculus, Description Logics and OWL DL.

Literal class

The resources for literal LV is named as the class extension of $rdfs:Literal^{\mathcal{I}}$.

$$LV \quad = \quad CEXT^{\mathcal{I}}(rdfs:Literal^{\mathcal{I}}) \tag{2.7}$$

As well as rdfs:Resource, rdfs:Literal also belongs to the universe of discourse.

$$rdfs:Literal^{\mathcal{I}} \quad \in \quad \boldsymbol{R}^{\mathcal{I}} \tag{2.8}$$

Universal metaclass

With respect to $CEXT^{\mathcal{I}}$ we considered $\boldsymbol{C}^{\mathcal{I}}$, which is the set of all classes in an interpretation $\mathcal{I}$. Lastly for naming several basic classes in the universe, the universal metaclass is introduced by naming the set of all classes defined in the universe, using new terminology ‘rdfs:Class’.

$$\boldsymbol{C}^{\mathcal{I}} \quad = \quad CEXT^{\mathcal{I}}(rdfs:Class^{\mathcal{I}}) \tag{2.9}$$

[8] Pat Hayes explained in his email `http://lists.w3.org/Archives/Public/www-tag/2007Sep/0168.html`, “The class of which all classes are subclasses is the universal class, which contains everything. Also called the ’universe’, also sometimes called the ’domain of discourse’, which draws attention to the fact that ’anything’ here means anything that can be referred to or talked about, whether it is real or imaginary: any possible topic of any kind of meaningful discourse.”

As well as $\boldsymbol{P}^{\mathcal{I}}$ and LV do, $\boldsymbol{C}^{\mathcal{I}}$ also turns out a subset of the resource in the universe. Namely,

$$\boldsymbol{C}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}. \tag{2.10}$$

Simultaneously, as well as $rdf:Property^{\mathcal{I}}$ and $rdfs:Literal^{\mathcal{I}}$ do, $rdfs:Class^{\mathcal{I}}$ itself turns out a resource in the universe. Namely,

$$rdfs:Class^{\mathcal{I}} \in \boldsymbol{R}^{\mathcal{I}}. \tag{2.11}$$

In addition, $rdfs:Class^{\mathcal{I}}$ is also in $\boldsymbol{C}^{\mathcal{I}}$, because $rdfs:Class^{\mathcal{I}} \in \boldsymbol{R}^{\mathcal{I}}$ and $\boldsymbol{C}^{\mathcal{I}}$ is the set of all classes in the universe.

$$rdfs:Class^{\mathcal{I}} \quad \in \quad \boldsymbol{C}^{\mathcal{I}} = CEXT^{\mathcal{I}}(rdfs:Class^{\mathcal{I}}). \tag{2.12}$$

Note that the universal class and the universal metaclass has the membership loop, (2.6) and (2.12), respectively. These membership loop caused strong alienation from RDF and RDFS semantic theory in a part of Semantic Web community, although Patrick Hayes and Brian McBride preventively stated in the document of RDF Semantics [25] as "In particular, this use of a class extension mapping allows classes to contain themselves. For example, it is quite OK for (the extension of) a 'universal' class to contain the class itself as a member, a convention that is often adopted at the top of a classification hierarchy." Here, $rdfs:Resource^{\mathcal{I}}$ sits at the top of *class hierarchy*, and $rdfs:Class^{\mathcal{I}}$ exists at the top of *class orders*[9] in the universe. Chapter 6 discusses OWL Full theory and formally deduces the membership loop again, based on only RDF and RDFS semantic conditions.

Superclass and subclass relation in RDFS semantic conditions

As described above, a class extension is a set of a subset of entities in the universe. This fact leads us the notion of super-sub relationship of classes associated to the notion of super set and sub set in set theory. The class super-sub relationship is specified with terminology 'rdfs:subClassOf' as the inclusiveness of the class extensions of classes as follows. This concept of super-sub relationship of classes is called *subsumption*.

[9] In this dissertation, "class hierarchy" is used to designate the super-sub relationship in classification or *subsumption*. On the other hand, "class orders" is used to designate the class-metaclass relationship after Russell's type theory.

(Subsumption) If a pair of two entities is a member of property extensions of $rdfs{:}subClassOf^{\mathcal{I}}$, then the both entities are instances of $rdfs{:}Class^{\mathcal{I}}$ and the class extension of the predecessor in the pair is included by the class extension of the successor.

$$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \Rightarrow x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) \subseteq CEXT^{\mathcal{I}}(y^{\mathcal{I}}) \quad (2.13)$$

This condition may be called *weak subsumption*, because it is not '*if and only if*' construct[10].

The reflection and transitivity on rdfs:subClassOf relation is trivially true from the notion of sub set on the class extension.
(Reflection) rdfs:subClassOf is self-reflective.

$$x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \quad (2.14)$$

(Transitivity) rdfs:subClassOf is transitive.

$$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \wedge \langle y^{\mathcal{I}}, z^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \Rightarrow \langle x^{\mathcal{I}}, z^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \quad (2.15)$$

In RDF semantics, every class in the universe is a subclass of rdfs:Resource.
(Top) rdfs:Resource is the top class of class hierarchy.

$$x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x^{\mathcal{I}}, rdfs{:}Resource^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \quad (2.16)$$

These rdf:type and rdfs:subClassOf relations between rdfs:Resource and rdfs:Class is depicted in **Fig. 2.1** together with all other entities in the universe.

Domain and range constraints for property

The RDF class-instance system may appear to be similar to type systems of object-oriented programming languages. However, RDF language is, precisely, not object-oriented, rather property-oriented. "RDF differs from many such systems in that instead of defining a class in terms of the properties its instances may have, the RDF vocabulary description language describes properties in terms of the classes of resource to which they apply. This is the role of the domain and range

[10]Oppositely, the subsumption rule with '*if and only if*' construct may be called *strong subsumption*. In OWL DL semantics, the strong subsumption is used. See Section 2.2 for the detail.

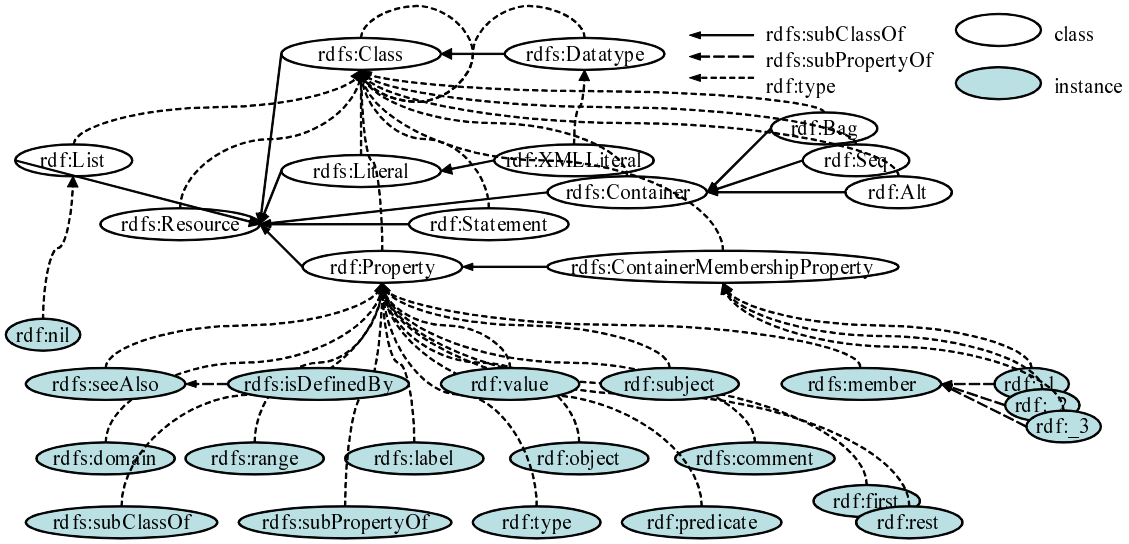

Fig. 2.1: RDF Resources and their Relations.

mechanisms described in this specification." (RDFSchema [7]) Using this RDF approach, it is easy for ones to subsequently define additional properties onto an ontology that is made by other people. It can be done without the need to redefine the original description of these classes. "One benefit of the RDF property-centric approach is that it allows anyone to extend the description of existing resources". (RDFSchema [7])

As described above, a property is a resource in the universe. Thus, a property may have its own properties, namely domain and range constraints in the case of RDFS.

(Domain) A subject in triple must satisfy the class restriction that is defined as domain constraint on the property.

$$\langle x^I, y^I \rangle \in EXT^I(rdfs:domain^I) \wedge \langle u^I, v^I \rangle \in EXT^I(x^I) \Rightarrow u^I \in EXT^I(y^I) \tag{2.17}$$

(Range) An object in triple must satisfy the class restriction that is defined as range constraint on the property.

$$\langle x^I, y^I \rangle \in EXT^I(rdfs:range^I) \wedge \langle u^I, v^I \rangle \in EXT^I(x^I) \Rightarrow v^I \in EXT^I(y^I) \tag{2.18}$$

Superproperty and subproperty in RDFS

As the inclusiveness of the class extension settles the notion of super-sub relationship on class, the inclusiveness of the property extension settles the notion of super-sub relationship on property. This super-sub property supports the inheritance of domain and range attributes on property.

(PropSubsumption)

$$\langle x^I, y^I \rangle \in EXT^I(rdfs:subPropertyOf^I) \Rightarrow x^I \in \boldsymbol{P}^I \wedge y^I \in \boldsymbol{P}^I \wedge EXT^I(x^I) \subseteq EXT^I(y^I) \quad (2.19)$$

(PropReflection) rdfs:subPropertyOf is self-reflexive.

$$x^I \in \boldsymbol{P}^I \Rightarrow \langle x^I, x^I \rangle \in EXT^I(rdfs:subPropertyOf^I) \quad (2.20)$$

(PropTransitive) rdfs:subPropertyOf is transitive.

$$\langle x^I, y^I \rangle \in EXT^I(rdfs:subPropertyOf^I) \wedge \langle y^I, z^I \rangle \in EXT^I(rdfs:subPropertyOf^I) \Rightarrow \langle x^I, z^I \rangle \in EXT^I(rdfs:subPropertyOf^I) \quad (2.21)$$

RDF Graph and Monotonicity

In semantic theory, it is critical to determine whether or not an additional piece of knowledge changes the previous interpretation. In order to know meanings of a set of pieces of knowledge, we must know all pieces of knowledge including what is implicitly contained. With a given set of triples and its interpretation, if a new additional triple within the given vocabulary and semantic conditions does not change the interpretation, it can be phrased 'the given set of triples and its interpretation *entail* the new triple', and the new triple is called an *entailment*. For example, in the case that we have a given triple $\langle s,\ p,\ o \rangle$, even if a triple $\langle s,\ rdf:type,\ rdfs:Resource \rangle$, which is not explicitly stated so far, is added, it does not change the previous interpretation, because every entity in the universe is a member of class $rdfs:Resource$. Adding new triples within the given vocabulary and semantic conditions, however, can make an interpretation more complex, more precise, and richer, with preserving the previous interpretation holds. For example, adding $\langle s,\ rdf:type,\ o' \rangle$ such as $\langle o',\ rdfs:subClassOf,\ o \rangle$ for $\langle s,\ p,\ o \rangle$ makes the previous interpretation more detailed without contradicting the interpretation held before.

Furthermore, adding new tripes with a new vocabulary and new semantic conditions can extend

the interpretation with preserving the previous interpretation holds.

> "Given a set of RDF graphs, there are various ways in which one can 'add' information to it. Any of the graphs may have some triples added to it; the set of graphs may be extended by extra graphs; or the vocabulary of the graph may be interpreted relative to a stronger notion of vocabulary entailment, i.e. with a larger set of semantic conditions understood to be imposed on the interpretations. All of these can be thought of as an addition of information, and may make more entailments hold than held before the change. All of these additions are monotonic, in the sense that entailments which hold before the addition of information, also hold after it." (RDF Semantics [25])

In Semantic Web theory, knowledge must be formalized to meet the requirement of monotonicity of knowledge. Under this *monotonicity principle*, the primitive world that has less triples is extended more precisely and richer along with additional triples with/without new vocabulary and extended interpretation rules. Thus, rdf-vocabulary and rdf-interpretation are superimposed to the simple interpretation, and rdfs-vocabulary and rdfs-interpretation are superimposed to rdf-vocabulary and rdf-interpretation. As well, owl-vocabulary and owl-interpretation must be superimposed to rdfs-vocabulary and rdfs-interpretation.

Entailment rules in RDF(S)

As shown in several semantic conditions so far, there are many entailment rules in RDF and RDFS. All of them are summarized at Table 2.1 through Table 2.5, which are taken from RDF Semantics [25]. Note that "*uuu aaa xxx* ." in tables represents a triple, which is previously expressed $\langle uuu, aaa, xxx \rangle$ or $\langle uuu^I, xxx^I \rangle \in EXT^I(aaa^I)$. Whereas these rules might look to be trivially true, but it is, in reality, not trivial. It is a sort of *word to the wise*. We can use these rules in many ways. For example, **rdf4a**. **rdf4b**, and **rdf1** allow us to define uuu^I and vvv^I as a resource and aaa^I as a property. Such entailments are utilized in SWCLOS for forward reference of ontology along with the monotonicity principle. See Chapter 3.

2.2 OWL Semantics

OWL can facilitate more precise ontology description than RDFS. The OWL specifications contain many features and capabilities that are useful to describe Web ontologies. For example, using OWL, ontologists can explicitly describe more precise and complex concepts on equivalence, disjointness,

Table 2.1: Simple Entailment Rules in RDF

Rule name	If *E* contains	then add	note
se1	*uuu aaa xxx* .	*uuu aaa* _:*nnn* .	where _:*nnn* identifies a blank node allocated to *xxx* by rule se1 or se2.
se2	*uuu aaa xxx* .	_:*nnn aaa xxx* .	where _:*nnn* identifies blank node allocated to *uuu* by rule se1 or se2.

Table 2.2: Literal generalization rule

Rule name	If *E* contains	then add	note
lg	*uuu aaa lll* .	*uuu aaa* _:*nnn* .	where _:*nnn* identifies a blank node allocated to the literal *lll* by this rule.

Table 2.3: Literal instantiation rule

Rule name	If *E* contains	then add	note
gl	*uuu aaa* _:*nnn* .	*uuu aaa lll* .	where _:*nnn* identifies a blank node allocated to the literal *lll* by rule lg.

Table 2.4: RDF entailment rules

Rule name	If *E* contains	then add	note
rdf1	*uuu aaa yyy* .	*aaa* rdf:type rdf:Property .	
rdf2	*uuu aaa lll* .	_:*nnn* rdf:type rdf:XMLLiteral .	where _:*nnn* identifies a blank node allocated to lll by rule lg.

Table 2.5: RDFS entailment rules

Rule	If *E* contains	then add	note
rdfs1	*uuu aaa lll* .	_:*nnn* rdf:type rdfs:Literal .	where *lll* is a plane literal w/o ltag. where _:*nnn* identifies a blank node allocated to lll by rule lg.
rdfs2	*aaa* rdfs:domain *xxx* . *uuu aaa yyy* .	*uuu* rdf:type *xxx* .	domain constraint
rdfs3	*aaa* rdfs:range *xxx* . *uuu aaa vvv* .	*vvv* rdf:type *xxx* .	range constraint
rdfs4a	*uuu aaa xxx* .	*uuu* rdf:type rdfs:Resource.	resource membership
rdfs4b	*uuu aaa vvv* .	*vvv* rdf:type rdfs:Resource.	resource membership
rdfs5	*uuu* rdfs:subPropertyOf *vvv* . *vvv* rdfs:subPropertyOf *xxx* .	*uuu* rdfs:subPropertyOf *xxx* .	transitivity of properties
rdfs6	*uuu* rdf:type rdf:Property .	*uuu* rdfs:subPropertyOf *uuu* .	reflection of properties
rdfs7	*aaa* rdfs:subPropertyOf *bbb* . *uuu aaa yyy* .	*uuu bbb yyy* .	super property
rdfs8	*uuu* rdf:type rdfs:Class .	*uuu* rdfs:subClassOf rdfs:Resource.	top class
rdfs9	*uuu* rdfs:subClassOf *xxx* . *vvv* rdf:type *uuu* .	*vvv* rdf:type *xxx* .	subsumption
rdfs10	*uuu* rdf:type rdfs:Class .	*uuu* rdfs:subClassOf *uuu* .	reflection of classes
rdfs11	*uuu* rdfs:subClassOf *vvv* . *vvv* rdfs:subClassOf *xxx* .	*uuu* rdfs:subClassOf *xxx* .	transitivity of classes
rdfs12	*uuu* rdf:type rdfs:ContainerMembershipProperty .	*uuu* rdfs:subPropertyOf rdfs:member .	ordinal property
rdfs13	*uuu* rdf:type rdfs:Datatype .	*uuu* rdfs:subClassOf rdfs:Literal .	top datatype

intersection, and union of concepts. However, W3C specified OWL semantics in two ways, i.e., Direct Model-Theoretic Semantics [59] and RDF-Compatible Model-Theoretic Semantics [58], and then it is stated that Direct Model-Theoretic Semantics takes precedence over RDF-Compatible Model-Theoretic Semantics, if the description of specifications contains contradictions.

In this section, for the sake of the unification of RDF semantics and OWL semantics, OWL semantics is described according to Tarskian denotational model semantics as well as RDF Semantics [25], while the description in Direct Model-Theoretic Semantics and other OWL documents [65, 49] are referred to.

2.2.1 OWL in Denotational Semantics

OWL vocabulary

In the Direct Model-Theoretic Semantics [59], OWL vocabulary $\mathcal{V}$ consists of vocabularies for literals and for eight kinds of URI references as follows:

$$\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_C \cup \mathcal{V}_D \cup \mathcal{V}_I \cup \mathcal{V}_O \cup \mathcal{V}_{DP} \cup \mathcal{V}_{IP} \cup \mathcal{V}_{AP} \cup \mathcal{V}_{OP}$$

$\mathcal{V}_L$: literals, i.e., plane literals and typed literals

$\mathcal{V}_C$: class names, e.g., owl:Thing, owl:Nothing, etc.

$\mathcal{V}_D$: datatype names, e.g., xsd:string, xsd:integer, etc.

$\mathcal{V}_I$: individual names

$\mathcal{V}_O$: ontology names

$\mathcal{V}_{DP}$: data valued property names, e.g., vin:yearValue.

$\mathcal{V}_{IP}$: individual property names, e.g., owl:sameAs, etc.

$\mathcal{V}_{AP}$: annotation property names, e.g., owl:versionInfo, rdfs:label, rdfs:comment, etc.

$\mathcal{V}_{OP}$: ontology property names

Here, $\mathcal{V}_C$ and $\mathcal{V}_D$ are disjoint, and $\mathcal{V}_{DP}$, $\mathcal{V}_{IP}$, $\mathcal{V}_{AP}$, and $\mathcal{V}_{OP}$ are pairwise disjoint.

Note that in OWL DL, $\mathcal{V}_C$ and $\mathcal{V}_I$ are disjoint, but OWL Full does not distinguish them.

OWL interpretation

The denotational interpretation for OWL can be formalized as follows[11]:

1. A non-empty set $\boldsymbol{OT}^{\mathcal{I}}$ of entities such that $\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$, called OWL universe of $\mathcal{I}$. $\boldsymbol{OT}^{\mathcal{I}}$ is disjoint from LV.
2. A set $\boldsymbol{OP}_D^{\mathcal{I}}$ such that $\boldsymbol{OP}_D^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, called a set of denotations of datatype properties.
3. A set $\boldsymbol{OP}_I^{\mathcal{I}}$ such that $\boldsymbol{OP}_I^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, called a set of denotations of individual properties.
4. A set $\boldsymbol{OP}_A^{\mathcal{I}}$ such that $\boldsymbol{OP}_A^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, called a set of denotations of annotation properties.
5. A set $\boldsymbol{OP}_O^{\mathcal{I}}$ such that $\boldsymbol{OP}_O^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, called a set of denotations of ontology properties.
6. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_D^{\mathcal{I}}$ into the powerset of $\boldsymbol{OT}^{\mathcal{I}} \times LV$.
7. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_I^{\mathcal{I}}$ into the powerset of $\boldsymbol{OT}^{\mathcal{I}} \times \boldsymbol{OT}^{\mathcal{I}}$.
8. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_A^{\mathcal{I}}$ into the powerset of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$.
9. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_O^{\mathcal{I}}$ into the powerset of $\boldsymbol{O}^{\mathcal{I}} \times \boldsymbol{O}^{\mathcal{I}}$.
10. A mapping $L^{\mathcal{I}}$ from typed literals in $\mathcal{V}_L$ into LV.
11. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}_I$ into $\boldsymbol{OT}^{\mathcal{I}}$.

Note that the semantics of property extension $EXT^{\mathcal{I}}$ in OWL is the same as one in RDF (2.1). Namely, the definitions such that $\boldsymbol{OP}_D^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, $\boldsymbol{OP}_I^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, $\boldsymbol{OP}_A^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$, and $\boldsymbol{OP}_O^{\mathcal{I}} \subset \boldsymbol{P}^{\mathcal{I}}$ enable the interpretation coincident with RDF semantics along with the definition $\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$.

Proof. With applying (2.1) to $\boldsymbol{OP}_I^{\mathcal{I}}$ and $\boldsymbol{OP}_O^{\mathcal{I}}$, RDF simple interpretation 3 and OWL interpretation 7 and 9 yield $\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$ and $\boldsymbol{O}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$.

Class and instance relation in OWL

The semantics of membership in OWL is the same as RDF, namely, the definition (2.2) also holds in OWL. Thus, the definitions $\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$ implies that every entity in OWL is also a resource of the universe of discourse in RDF. Hereafter, let us call $\boldsymbol{OT}^{\mathcal{I}}$ OWL universe, and $\boldsymbol{R}^{\mathcal{I}}$ RDF universe. The OWL universe is a subset of RDF universe.

[11] These descriptions are derived as they simulate the way of formalization in RDF Semantics. However, the descriptions are almost same as the RDF-Compatible Model-Theoretic Semantics [58] but extended to contain the description of RDF universe.

Strong subsumption in OWL

While the super-sub relationship in RDF classes is *weak subsumption* condition (2.13), that in OWL universe is *strong subsumption* condition. (OWL Semantics [58], OWL2 [9] Table 5.8) See the following formula.

$$\langle x^I, y^I \rangle \in EXT^I(rdfs{:}subClassOf^I) \Leftrightarrow x^I \in \boldsymbol{OC}^I \wedge y^I \in \boldsymbol{OC}^I \wedge CEXT^I(x^I) \subseteq CEXT^I(y^I) \quad (2.22)$$

Where $\boldsymbol{OC}$ is a set of all classes in OWL universe. This formula means in the direction of the right hand side to the left hand side; if we find that a class extension $CEXT^I(C^I)$ of class C^I is a subset of a class extension $CEXT^I(D^I)$ of class D^I where C^I is not related to D^I on rdfs:subClassOf, C^I is not a subclass of D^I in RDF semantics but it is so in OWL semantics. In other words, if two classes are in super-sub relationship of classes of RDF semantics, then it is so in OWL semantics (from the left hand side to the right hand side), but not vice versa.[12]

This strong condition allows ones to decide that a class that has a class extension as a subset included another set is a subclass of another class that has a class extension as its superset, even though both classes are not connected in RDF graph. It also implies that the class notion in OWL is regarded the same as set. The super/subset relationship in set theory coincides with super/subclass relationship of OWL classes. Therefore, the intensional definition for OWL classes substantially coincides with the extensional definition of classes like set theory[13].

Top class and bottom class in OWL

The OWL universe $\boldsymbol{OT}^I$ is a powerset that contains all individuals in OWL, and then it is named as an extension of a class $owl{:}Thing^I$. On the other hand, we may have a bottom class as an empty set in the OWL universe, and it is named as an extension of $owl{:}Nothing^I$.

1. $CEXT^I(owl{:}Thing^I) = \boldsymbol{OT}^I$
2. $CEXT^I(owl{:}Nothing^I) = \emptyset$, empty set

[12] SWCLOS, which is object oriented OWL processor in OWL Full level, adopts the weak subsumption not only in RDF universe but also in OWL universe, whereas the complete relationship (iff relationship) for owl:intersectionOf, owl:unionOf, and owl:oneOf is adopted and computed in `gx:subsumed-p` in OWL universe.

[13] This condition is too strong for object-oriented languages that are coupled with OWL semantics. In this semantics, we cannot distinguish two distinct sets which contains twin objects that hold completely the same slot values. Therefore, the weak condition for subsumption on rdfs:subClassOf is preserved in SWCLOS.

Obviously by set theory, owl:Thing is a superclass of any other class, and owl:Nothing is a subclass of any other classes in OWL universe.

Class extension in OWL

Supposing a collection of all classes in OWL universe, namely $\boldsymbol{OC}^{\mathcal{I}}$, and name it as an class extension of $owl:Class^{\mathcal{I}}$. Note that the OWL universe is a subset of the RDF universe, $\boldsymbol{OT}^{\mathcal{I}} \subseteq \boldsymbol{R}^{\mathcal{I}}$, and the semantics of rdf:type and the membership is the same between RDF and OWL, then it is derived that every class in OWL universe belongs to RDF universe, namely $\boldsymbol{OC}^{\mathcal{I}} \subseteq \boldsymbol{C}^{\mathcal{I}}$.

Specialized interpretation for OWL universe

The followings are a specialized version of RDF simple interpretation for OWL universe.

1. A non-empty set $\boldsymbol{OC}^{\mathcal{I}}$ such that $\boldsymbol{OC}^{\mathcal{I}} \subseteq \boldsymbol{C}^{\mathcal{I}}$, called a set of denotations of OWL classes.
2. A mapping $CEXT^{\mathcal{I}}$ from $\boldsymbol{OC}^{\mathcal{I}}$ into the powerset of $\boldsymbol{OT}^{\mathcal{I}}$.
3. A mapping $CEXT^{\mathcal{I}}$ from $\boldsymbol{OD}^{\mathcal{I}}$ into the powerset of LV.
4. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}_C$ into $\boldsymbol{OC}^{\mathcal{I}}$.
5. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}_D$ into $\boldsymbol{OD}^{\mathcal{I}}$.
6. $CEXT^{\mathcal{I}}(owl:Class^{\mathcal{I}}) = \boldsymbol{OC}^{\mathcal{I}}$

Class property in OWL

It might sound strange but any property extension of classes for class vocabulary $\mathcal{V}_C$ is not defined in the Direct Model-Theoretic Semantics [59]. The reason is that the relationship among classes falls into a category of higher order logic. The Direct Model-Theoretic Semantics is for OWL DL, which is based on Description Logic, a subset of first order logic. Therefore, the extension of classes for class vocabulary goes beyond the scope of first order logic and DL. We introduce here a vocabulary for class property $\mathcal{V}_{CP}$ and the notion of property extension for classes $\boldsymbol{OP}_C^{\mathcal{I}}$ as well as the vocabulary for individual property $\mathcal{V}_{IP}$ and the property extension for individuals $\boldsymbol{OP}_I^{\mathcal{I}}$[14]

[14] In case that we consider owl:intersectionOf or owl:unionOf are words in $\mathcal{V}_{CP}$, the definition of $\boldsymbol{OP}_C^{\mathcal{I}}$ does not match the discussion in the RDF-Compatible Model-Theoretic Semantics [58], because those class properties take a sequence of classes and every range of them is rdf:List. However, rdf:List as construct of sequence of classes is problematic so that it makes it easy to fall into a trap of misunderstanding 'comprehension principle'. In this section, we stay clear of disputes on rdf:List for sequences of classes. See the detailed discussion in Chapter 6.

1. A set $\boldsymbol{OP}_C^{\mathcal{I}}$ such that $\boldsymbol{OP}_C^{\mathcal{I}} \subseteq \boldsymbol{P}^{\mathcal{I}}$.
2. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_C^{\mathcal{I}}$ into the powerset of $\boldsymbol{OC}^{\mathcal{I}} \times \boldsymbol{OC}^{\mathcal{I}}$.

Apart from the restriction that arises from first order logic, the relationship around individuals and classes in OWL universe are emerging in homo-morphological image between the interpretation for individuals and the interpretation for classes in the front of our eyes. The following conditions are the result of rearrangement of those conditions mentioned above.
For individuals;

1. A non-empty set $\boldsymbol{OT}^{\mathcal{I}}$ of entities such that $\boldsymbol{OT}^{\mathcal{I}} \subseteq \boldsymbol{R}^{\mathcal{I}}$.
2. A set $\boldsymbol{OP}_I^{\mathcal{I}}$ such that $\boldsymbol{OP}_I^{\mathcal{I}} \subseteq \boldsymbol{P}^{\mathcal{I}}$.
3. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_I^{\mathcal{I}}$ into the powerset of $\boldsymbol{OT}^{\mathcal{I}} \times \boldsymbol{OT}^{\mathcal{I}}$.
4. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}_I$ into $\boldsymbol{OT}^{\mathcal{I}}$.

For classes;

1. A non-empty set $\boldsymbol{OC}^{\mathcal{I}}$ such that $\boldsymbol{OC}^{\mathcal{I}} \subseteq \boldsymbol{C}^{\mathcal{I}}$.
2. A set $\boldsymbol{OP}_C^{\mathcal{I}}$ such that $\boldsymbol{OP}_C^{\mathcal{I}} \subseteq \boldsymbol{P}^{\mathcal{I}}$.
3. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{OP}_C^{\mathcal{I}}$ into the powerset of $\boldsymbol{OC}^{\mathcal{I}} \times \boldsymbol{OC}^{\mathcal{I}}$.
4. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}_C$ into $\boldsymbol{OC}^{\mathcal{I}}$.

These homo-morphological semantic conditions between individuals and classes ensure a context dependent subsumption algorithm. Namely, the same algorithm does work well for both individuals (and to their classes) and classes (and to their metaclasses). Thus, it enables *punning* in context and the object oriented method programming, of which program code will work for individuals with their classes and classes with their metaclasses.

Equivalent OWL classes

In OWL, the meaning of the property extension of owl:equivalentClass is that the two classes described have exactly the same class extension. See the following definition.

$$\langle x^{\mathcal{I}},\, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(owl\!:\!equivalentClass^{\mathcal{I}}) \Leftrightarrow$$
$$x^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) = CEXT^{\mathcal{I}}(y^{\mathcal{I}})$$

Complement of concept

Among OWL properties, three properties, i.e., owl:complementOf, owl:intersectionOf, and owl:unionOf make a complete relation between the left-hand side and the right-hand side without using owl:equivalentClass. The description with owl:complementOf states that a class extension of one class is complement of another in OWL universe.

$$\langle x^{\mathcal{I}},\, y^{\mathcal{I}}\rangle \in EXT^{\mathcal{I}}(owl{:}complementOf^{\mathcal{I}}) \Leftrightarrow$$
$$x^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) = \boldsymbol{OT} \setminus CEXT^{\mathcal{I}}(y^{\mathcal{I}})$$

Intersection and union of concepts

Both owl:intersectionOf and owl:unionOf take a collection of classes as their object in triple, and owl:intersectionOf makes a subconcept and owl:unionOf makes superconcept of collection members.

$$\langle x^{\mathcal{I}},\, \{y_1^{\mathcal{I}}, y_2^{\mathcal{I}},\, \ldots,\, y_n^{\mathcal{I}}\}\rangle \in EXT^{\mathcal{I}}(owl{:}intersectionOf^{\mathcal{I}}) \Leftrightarrow$$
$$x^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \bigwedge_{i=1,\ldots,n} y_i^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) = \bigcap_{i=1,\ldots,n} CEXT^{\mathcal{I}}(y_i^{\mathcal{I}}) \quad (2.23)$$

$$\langle x^{\mathcal{I}},\, \{y_1^{\mathcal{I}}, y_2^{\mathcal{I}},\, \ldots,\, y_n^{\mathcal{I}}\}\rangle \in EXT^{\mathcal{I}}(owl{:}unionOf^{\mathcal{I}}) \Leftrightarrow$$
$$x^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \bigwedge_{i=1,\ldots,n} y_i^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) = \bigcup_{i=1,\ldots,n} CEXT^{\mathcal{I}}(y_i^{\mathcal{I}}) \quad (2.24)$$

Note that this entailment rules are modified from RDF and OWL semantics written in W3C recommendations. The domain constraint of owl:intersectionOf and owl:unionOf is owl:Class, but the range constraint is rdf:List. However, this range constraint is problematic so that it tends to confuse the interpretation of collection or rdf:List in the discussion of the RDF compatibility of OWL DL. Therefore, we do not use rdf:first and rdf:rest in order to embody a collection. Instead we make a list in Lisp and handle directly a list and its members in SWCLOS. This interpretation basically does not contradict to form an RDF graph as collection of classes that compatible to a sequence of classes. The details with respect to *comprehension principle* and collections for owl:intersectionOf and owl:unionOf are discussed in Chapter 6.

Disjoint classes

In OWL, the disjoint relationship of two classes may be explicitly described as follows.

$$\langle x^I,\ y^I\rangle \in EXT^I(owl{:}disjointWith^I) \Rightarrow x^I \in \boldsymbol{OC}^I \wedge y^I \in \boldsymbol{OC}^I \wedge CEXT^I(x^I) \cap CEXT^I(y^I) = \emptyset$$

The discussion on disjoint classes is problematic. See the discussion at Chapter 7.

OneOf property

owl:oneOf makes axiomatic complete assertions.

$$\langle x^I,\ \{y_1^I,\ y_2^I,\ \ldots,\ y_n^I\}\rangle \in EXT^I(owl{:}oneOf^I) \Leftrightarrow$$
$$x^I \in \boldsymbol{C}^I \bigwedge_{i=1,\ldots,n} y_i^I \in \boldsymbol{R}^I \wedge CEXT^I(x^I) = \{y_1^I,\ y_2^I,\ \ldots,\ y_n^I\}$$

Individuals and non-Unique Name Assumption

According to the Tarskian denotational semantics, the semantics of owl:sameAs and owl:differentFrom can be formalized as follows.
(owl:sameAs) If x and y are different URIs and the both references make a pair that is an extension of $owl{:}sameAs^I$, then the denotation of x and y are the same one.

$$\{x \neq y \mid x \in \mathcal{V}_I,\ y \in \mathcal{V}_I\} \wedge \langle x^I,\ y^I\rangle \in EXT^I(owl{:}sameAs^I) \Rightarrow x^I = y^I \tag{2.25}$$

(owl:differentFrom) If x and y are different URIs and the both references make a pair that is an extension of $owl{:}differentFrom^I$, then the denotation of x and y are different.

$$\{x \neq y \mid x \in \mathcal{V}_I,\ y \in \mathcal{V}_I\} \wedge \langle x^I,\ y^I\rangle \in EXT^I(owl{:}differentFrom^I) \Rightarrow x^I \neq y^I \tag{2.26}$$

These conditions are called *non-Unique Name Assumption*. It is, however, unusual for predicate calculus, programming languages, and other most mathematical branches and computer applications, even though it is a common usage like nicknames in everyday languages.

The non Unique Name Assumption implies that we need to decide the sameness of denotations of symbols or names in order to obtain useful results from knowledge or ontologies. There are two basic questions arising. Firstly, what we should do in case that we do not have sufficient information to decide the sameness or differentness of denotations. Secondly, different URIs denote different

graph nodes in RDF semantics. So, how we should treat owl:sameAs and owl:differentFrom in RDF semantics and in the combination of RDF semantics and OWL semantics, namely OWL Full. Section 6.4 discusses these questions and addresses a solution that is implemented in SWCLOS.

Property restriction

The domain constraint and range constraint of property in RDF is global upon a property, and they restrict the subject class and the object class of a predicate in a triple. Therefore, the range constraint has an effect for any subject of triples. On the other hand, property restrictions in OWL are applied to subjects to which and predicates on which owl:onProperty is designated. There are three kinds of value constraints and three kinds of cardinality constraints.

$$\langle x^I, y^I \rangle \in EXT^I(owl:allValuesFrom^I) \wedge \langle x^I, p^I \rangle \in EXT^I(owl:onProperty^I) \Rightarrow \{u^I \in \boldsymbol{OT}^I \mid \langle u^I, v^I \rangle \in EXT^I(p^I) \rightarrow v^I \in CEXT^I(y^I)\} \tag{2.27}$$

$$\langle x^I, y^I \rangle \in EXT^I(owl:someValuesFrom^I) \wedge \langle x^I, p^I \rangle \in EXT^I(owl:onProperty^I) \Rightarrow \{u^I \in \boldsymbol{OT}^I \mid \exists \langle u^I, v^I \rangle \in EXT^I(p^I) \wedge v^I \in CEXT^I(y^I)\}$$

$$\langle x^I, y^I \rangle \in EXT^I(owl:hasValue^I) \wedge \langle x^I, p^I \rangle \in EXT^I(owl:onProperty^I) \Rightarrow \{u^I \in \boldsymbol{OT}^I \mid \exists \langle u^I, y^I \rangle \in EXT^I(p^I)\}$$

$$\langle x^I, n \rangle \in EXT^I(owl:minCardinality^I) \wedge \langle x^I, p^I \rangle \in EXT^I(owl:onProperty^I) \Rightarrow \{u^I \in \boldsymbol{OT}^I \mid \ \mid \{v^I \in \boldsymbol{OT}^I \cup LV\} \wedge \langle u^I, v^I \rangle \in EXT^I(p^I) \mid \ \geq n\}$$

$$\langle x^I, n \rangle \in EXT^I(owl:maxCardinality^I) \wedge \langle x^I, p^I \rangle \in EXT^I(owl:onProperty^I) \Rightarrow \{u^I \in \boldsymbol{OT}^I \mid \ \mid \{v^I \in \boldsymbol{OT}^I \cup LV\} \wedge \langle u^I, v^I \rangle \in EXT^I(p^I) \mid \ \leq n\}$$

$$\langle x^{\mathcal{I}}, n\rangle \in EXT^{\mathcal{I}}(owl{:}cardinality^{\mathcal{I}}) \wedge \langle x^{\mathcal{I}}, p^{\mathcal{I}}\rangle \in EXT^{\mathcal{I}}(owl{:}onProperty^{\mathcal{I}}) \Rightarrow$$
$$\{u^{\mathcal{I}} \in \boldsymbol{OT}^{\mathcal{I}} \mid \ |\{v^{\mathcal{I}} \in \boldsymbol{OT}^{\mathcal{I}} \cup LV\} \wedge \langle u^{\mathcal{I}}, v^{\mathcal{I}}\rangle \in EXT^{\mathcal{I}}(p^{\mathcal{I}})| \ = n\}$$

2.2.2 OWL Entailment Rules

Since OWL provides many facilities for ontology description [49] and the combination of these features bears many numbers of entailment rules, all of possible entailment rules are not exhausted. Some of them are disclosed by ter Horst [72]. Table 2.6 lists a set of OWL entailment rules by ter Horst. Note that additional rules are found by us (See Table 3.3) and implemented into SWCLOS together with ones by ter Horst.

2.3 CLOS Semantics

Common Lisp [68, 69] is a dialect of lisp languages, which is produced by the activity of ANSI standardization on lisp dialects in U.S. Many applications are programmed in Common Lisp languages and they are running from academia to industry today. Common Lisp Object System (CLOS) is an object oriented extension of Common Lisp, which was born as a result of the work of introducing an object oriented programming paradigm into Common Lisp by unifying mainly two object oriented languages at that time, Common Loops and Flavors, which ran on two major lisp machines. The CLOS specification has been integrated to Common Lisp in ANSI Common Lisp version 2 [69].

Note that Common Lisp is a specification of a list language, and not a name for actual language systems. There are several Common Lisp systems from open source programs to commercial-based systems.

2.3.1 CLOS View of Object Oriented Programming

Compared to today's major OOP languages like Java or C#, CLOS has unique features. The fundamentals of CLOS are *classes*, *instances*, *generic functions*, and *methods* [69].

Class and instance. A class in CLOS is an object that determines the structure and behavior of other objects, namely its instances. Every object in CLOS is an instance of a class. Thus, any class object is also an instance of another class.

Table 2.6: OWL Entailment Rules

Rule	If E contains	then add	note
rdfp1	*p* type FunctionalProperty . *u p v* . *u p w* .	*v* sameAs *w* .	functional property
rdfp2	*p* type InverseFunctionalProperty *u p w* . *v p w* .	*u* sameAs *v* .	inverse functional property
rdfp3	*p* type SymmetricProperty . *v p w* .	*w p v* .	symmetric property
rdfp4	*p* type TransitiveProperty . *u p v* . *v p w* .	*u p w* .	transitive property
rdfp5a	*v p w* .	*v* sameAs *v* .	self-evident sameAs
rdfp5b	*v p w* .	*w* sameAs *w* .	
rdfp6	*v* sameAs *w* .	*w* sameAs *v* .	reflective sameAs
rdfp7	*u* sameAs *v* . *v* sameAs *w* .	*u* sameAs *w* .	transitive sameAs
rdfp8ax	*p* inverseOf *q* . *v p w* .	*w q v* .	inverse of property
rdfp8bx	*p* inverseOf *q* . *v q w* .	*w p v* .	symmetric inverse
rdfp9	*v* type Class . *v* sameAs *w* .	*v* subClassOf *w* .	subsumption on same classes
rdfp10	*p* type Property . *p* sameAs *q* .	*p* subPropertyOf *q* .	subsumption on same properties
rdfp11	*u p v* . *u* sameAs *u′* . *v* sameAs *v′* .	*u′ p v′* .	property extension through sameAs
rdfp12a	*v* equivalentClass *w* .	*v* subClassOf *w* .	subsumption on equivalentClass
rdfp12b	*v* equivalentClass *w* .	*w* subClassOf *v* .	
rdfp12c	*v* subClassOf *w* . *w* subClassOf *v* .	*v* equivalentClass *w* .	equivalency from subsumption classes
rdfp13a	*v* equivalentPropety *w* .	*v* subPropertyOf *w* .	subsumption on equivalent property
rdfp13b	*v* equivalentPropety *w* .	*w* subPropertyOf *v* .	*ditto*
rdfp13c	*v* subPropertyOf *w* . *w* subPropertyOf *v* .	*v* equivalentProperty *w* .	equivalency from subsumption properties
rdfp14a	*v* hasValue *w* . *v* onProperty *p* . *u p w* .	*u* type *v* .	filler restriction entailment
rdfp14bx	*v* hasValue *w* . *v* onProperty *p* . *u* type *v* .	*u p w* .	hasValue definition
rdfp15	*v* someValuesFrom *w* . *v* onProperty *p* . *u p x* . *x* type *w* .	*u* type *v* .	full existential restriction entailment
rdfp16	*v* allValuesFrom *w* . *v* onProperty *p* . *u* type *v* . *u p x* .	*x* type *w* .	allvaluesfrom definition

Generic function and method. A generic function is an extension of function whose behavior depends on the classes of the arguments supplied to it. A generic function is an aggregation or abstraction of specialized methods that have an identical name and the same number of function parameters but each method has the parameters associated to different classes. An actual method to be invoked is determined at run time by matching classes specified for parameters against actual classes of arguments supplied to the parameters. This method invocation mechanism is called *method dispatching*. The object oriented paradigm in CLOS is based on this method dispatching mechanism rather than on message-passing to objects that is invented by Smalltalk. It facilitates flexible context dependent programming. CLOS does not provide the facility of the encapsulation of object data.

Multiple inheritance. The structure and behavior of objects are inherited from superclasses of the class of an object. Whereas classes in Java and C# has a single direct superclass to inherit the structure and behavior, a CLOS class may have more than one direct superclass, and an object can inherit the structure and behavior from all of multiple superclasses of the class.

Method combination. In CLOS standard method combination, there are four kinds of methods, *primary*, *before* , *after* and *around method*. The before method is a sort of prologue for the primary method in order to make preparations, and the after method is a sort of epilogue that performs housekeeping or clearing. The around method can change values of supplied arguments before primary method invocation. The most specific primary method in the combination of argument's classes among methods matched by method dispatching is invoked, and a programmer can encode in primary methods to pass a part of work to more abstract methods or delegate it to another method.

In addition to the standard method combination, a programmer can change the method combination strategy from the standard one to another one.

First-class objects. In CLOS, generic functions and classes are first-class objects. Namely, they can be arguments for function calling. It is possible to create an anonymous function (lambda function) and anonymous class, and then they can be manipulated as object in the system as well as named objects.

2.3.2 Class Based System and Miscellaneous

CLOS is a class based system. A class in CLOS is a mother prototype of its instances, and inevitably an instance belongs to only one direct class. A class must be defined before making its instance.

Class precedence list. In CLOS realization, all superclasses of a class are collected and cached into the slot named `mop:class-precedence-list`. The value of class precedence lists of a class are computed at the time of the first creation of its instance.

Slot inheritance. Each class has its own slot definitions for instances. It is called *direct slot definition*. Each class also has the slot definitions that are inherited from all superclasses of the class. It is called *effective slot definitions*. The effective slot definitions provides a source of making slots of instances. [31]

Slot value type option. The direct slot definition and the effective slot definition can hold a specifier of class constraint for slot value. It can play a role of constraint of slot value in instance. The slot definition inheritance mechanism is equipped with the mechanism of retrieving and accumulating the inherited slot value type option.

Changing a class of instance. CLOS provides the functionality of changing a class of an instance at run time. The slot structure of the instance may be modified by changing the class.

Redefinition of class. CLOS provides the functionality of redefining a class at run time. It involves redefining the slot structure of instance objects. Redefining a class causes the effects on its all instances. The effect of redefining class automatically propagates to its subclasses and instances of subclasses. A programmer can control this propagation.

2.3.3 Meta-circularity in CLOS and Meta-Object Protocol

The most remarkable feature of CLOS is the openness of language specification for users. “The basic idea of the CLOS design is to specify a model for the language implementation and to standardize it. The inner workings of the implementation thereby become manipulable in a controlled manner. This internal model is called the CLOS Metaobject Protocol (MOP).” (Andreas Paepcke [56]) CLOS itself is written in CLOS. This unity of language is called *meta-circularity* of language. It is usual way in computer languages. The compiler of C is programmed in C, and the compiler of Common

Lisp is programmed in Common Lisp. The feature of CLOS is that it is opened to programmers through MOP. Users of CLOS can tailor the specification of CLOS using MOP. Ordinary object oriented languages like Java or C# are not equipped with MOP, the compiling machinery of classes and methods are ordinarily built in compiler systems and closed to users, although modern object oriented languages have recently become to provide the way of retrieving class information of objects at run time by the name of 'reflection'. However, precisely "The ability to modify the language's implementation without leaving the realm of the languages is called *reflection*." (Andreas Paepcke [56])

Generally speaking, the following conditions are required for any computational systems to be reflective. [64][46].

1. A computational system represents and exposes the components and the state of itself and they are manipulable.

2. There is a *causality* between the target system for computation and the computational system. Thus, the results of computation in one system level may propagate to other levels.

3. Inevitably, the reflective system must be infinitely layered so that a target system is controlled by the computational system, then the computational system must be realized and controlled another system, and then the system for the computational system must be realized by another system, $\cdots$ and so forth. Thus, some trick is needed to suppress this infinite layering into finite machinery.

These requirements for reflective system are realized as follows for reflective object oriented language CLOS.

1. Every class in CLOS is realized as an object in CLOS, and it is manipulated through CLOS standardized Meta-Object Protocol for metaobjects. [31].

2. There is no difference in semantics and realizations among built-in metaobjects for CLOS realization and user defined metaobjects for target systems, and they exist in the unity of system.

3. The class of cl:standard-class is cl:standard-class itself (meta-circularity and membership-loop), whereby infinite meta-circularity in reflective object systems is virtually realized.

Where the following demonstration shows the meta-circularity of systems by the membership loop of cl:standard-class.

```
(defclass C1 (standard-class) ()
  (:metaclass standard-class))
-> #<standard-class C1>
(defclass C2 (standard-class) ()
  (:metaclass C1))
-> #<C1 C2>
(defclass C3 (standard-class) ()
  (:metaclass C2))
 -> #<C2 C3>
```

In this simple demonstration, note that C3 is a metaclass at the same level of cl:standard-class, C2 has become to be a meta-metaclass at the same level of cl:standard-class, C1 has become to be a meta-meta-metaclass at the same level of cl:standard-class. Notably, the membership loop at cl:standard-class enabled such meta-level layering among C3, C2, and C1. Note that the first definition on C1 does not designate it as a meta-meta-metaclass. It is defined as meta-class firsly, but it is forced to play a role of meta-metaclass when C2 is defined, and it is also forced to play a role of meta-meta-metaclass when C3 is defined.

All slot definitions and methods defined at cl:standard-class can be inherited to class C3, C2, and C1, and methods can be modified and specialized differently at each class at each meta-level. Thus, the standard specification of CLOS which is implemented at cl:standard-class can be tailored using this Meta-Object Protocol that is enabled by the membership loop of cl:standard-class. Also note that cl:standard-object is a superclass of cl:standard-class and it is an instance of cl:standard-class. Namely, ordinary classes as subclass of cl:standard-object can be treated as instance of cl:standard-class, and simultaneously slots and behavior of cl:standard-object are inherited to cl:standard-class and its subclasses. It is critical to know the relationship between rdfs:Resource and rdfs:Class in RDF(S) is analogous to such relationship between cl:standard-object and cl:standard-class in CLOS. We call this specific relationship *twisted*, and it is formally discussed in Chapter 6

2.3.4 Computational Models of Lisps

In this subsection, the category of models on computer languages are discussed with emphasizing the specialty of lisps as computer language, according to the semantics that was addressed by Brian Smith for the first reflective lisp system 3-lisp [64].

In a lisp system like early lisp system Lisp 1.5, which is equipped with symbol, function, and list, and without any other structural devices like object in object oriented programming, a syntactic lisp expression (S-expression) is reduced to a syntactic nominal form that is semantically equivalent to the original non-nominal form in λ-calculus. For example, an expression "`(+ 1 2)`" is reduced to

"3". However, we can quote an expression to suppress the reduction, and then we can change expressions and construct another lisp form, such as "`(cons (quote *) (cdr (quote (+ 1 2))))`" produces "`(* 1 2)`". This specific feature of lisp family languages is recently called *homoiconic*. In this first computational model, lexical expressions are not discriminated from the denotations. In other words, we have no notion of denotation. **Fig. 2.2** pictures this model. In the figure, the mapping from notations to objects is carried out in human brains. The reduction ψ in lambda calculus or entailment ψ in logic must follow the relationship of objects in the world. The theory of ordinary predicate calculus and set theories falls into this computational model.

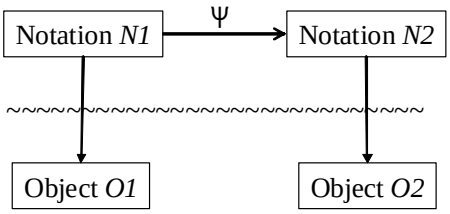

Fig. 2.2: The First Computation Model.

The second computational model discriminates symbols in expression from the denotations. A lexical expression "3" denotes number 3, and a lexical expression "t" and "nil", which are reduced to "t" and "nil", respectively, denote true and falsity, respectively, in the universe of discourse. However, an undefined symbol denotes nothing until it is defined in lisp systems among the network of denotations. Namely, we recognize the denotations as some computational object in a lisp system. See **Fig. 2.3**. This mapping ϕ from a lexical expression to the denotation as computational object is analogous to that in RDF semantics, in which a URI reference denotes a node in RDF graph. In this model, a denotation denotes an object in the world, and then rules that follow the world must be superimposed onto the reduction or entailment rules ψ.

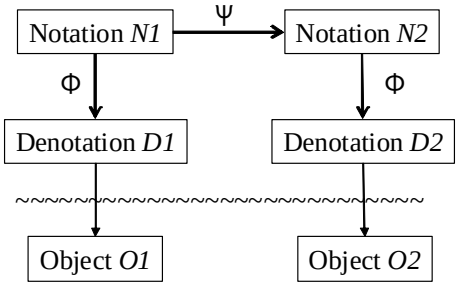

Fig. 2.3: The Second Computation Model.

In the third computational model, a symbol refers a structure as internal structured device in a modern computer language as well as the external list structure in the first lisp model. In this case, a symbol can be used to refer a referent that denotes an entity in the universe. See **Fig. 2.4**.

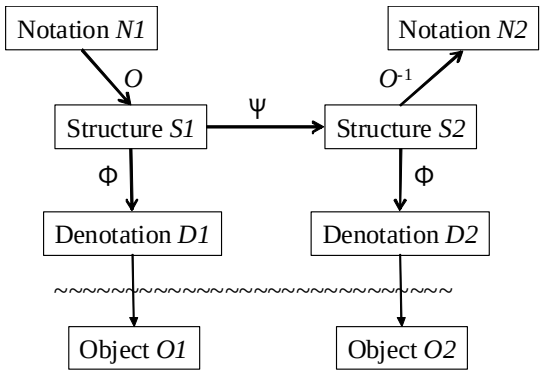

Fig. 2.4: The Third Computation Model.

Brian Smith called the mapping O *internalization*, and the inverse operation O^{-1} *externalization*. [64] He also noted that O (and O^{-1}) is usually ignored in logic. The ϕ is the interpretation function, which is analogous to the interpretation in denotational semantics, and the reduction ψ, which is, Smith says, the relationship among symbols, corresponds entailment rules and rule application in logic. Smith pointed out that the lisp evaluator crosses semantical levels, and therefore obscures the difference between the simplification ψ and the interpretation ϕ. Smith called this lisp specific nature *de-reference* ($\phi = \psi$). It has become the theoretic base of his work on the reflective language 3-Lisp.

Assumptions and axioms in domain knowledge can be syntactically represented by a set of symbols and structures expressed among them. Those expressions of assumptions are reduced to entailed assertions by the entailment rules ψ. It must follow rules in the world.

The model of SWCLOS is appropriate to this third model. Every URI reference or QName is internalized to a CLOS object. The CLOS objects for RDF embody nodes in RDF graph by RDF semantics. The ψ is organized to match RDF(S) entailment rules. RDFS subsumption rule is almost same as CLOS subsumption rule. OWL entailment rules must be superimposed to RDF(S) entailment rules ψ such follow the rules in the world.

2.4 Semantic Gaps between OWL and OOPLs

The Software Engineering Task Force (SETF) of W3C compared OWL features with ordinary Object-Oriented Programming Languages (OOPLs), and pointed out the serious semantic discrepancies as **Table 2.7** [35]. Note that here the class in OOPLs is compared to the OWL class, and the instance in OOPLs is compared to the OWL individual. The slot or the member variable of objects corresponds to the property and its value (role and filler) in OWL.

Nevertheless, some of these items are not issued to dynamic OOPLs such as CLOS as described above. The dynamic features of CLOS are summarized as follows.

Multiple Class Inheritance: Methods and slot definitions are inherited from multiple classes.[15]

Dynamic Programming: CLOS provides the means to redefine a class at runtime.

Metaobject: A class in CLOS is the first-class entity as object; thus a class in CLOS is called *metaobject*.

Metaclass: A metaclass or a class of classes allows ones to modify methods for classes including intrinsic methods in the system by using MOP. Specifically, the instance creation method is customizable in CLOS through metaclasses.

Reflective Programming: A programmer can alter behaviors of CLOS system using MOP.

Thus, it was plausible to implement RDF and OWL semantics leveraging CLOS dynamic features and MOP.

2.5 Concluding Remarks

In this chapter, firstly, RDF semantics are overviewed according to the description of RDF Semantics [25] of W3C Recommendation. The brief explanation on Tarskian denotational semantics is also presented. Secondly, OWL semantics is formalized in Tarskian denotational semantics as well as RDF Semantics, with focusing mainly membership and subsumption of entities. The semantic disparity between RDF(S) and OWL on membership and subsumption are made clear in detail. Due to no formalization so far on OWL semantics by Tarskian denotational semantics, this part is our original contribution for Semantic Webs. The semantic similarity between RDFS and CLOS is also

[15]This item does not relate to RDF(S) and OWL semantics directly but indirectly relates to the implementation of multiple classing in SWCLOS, See Subsection 3.1.6

Table 2.7: A Comparison of OWL/RDF and Object-Oriented Languages (by SETF [35])

Object-Oriented Languages	**OWL and RDF**
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values(literals) as values.	
Classes and Instances	
Classes are regarded as types for instances.	Classes are regarded as sets of individuals.
Each instance has one class as its type. Classes cannot share instances.	Each individual can belong to multiple classes.
Instances can not change their type at runtime.	Class membership may change at runtime.
The list of classes is fully known at compile-time and cannot change after that.	Classes can be created and changed at runtime.
Compilers are used at build-time. Compile-time errors indicate problems.	Reasoners can be used for classification and consistency checkingat runtime or build-time.
Properties, Attributes and Values	
Properties are defined locally to a class (and its subclasses through inheritance).	Properties are stand-alone entities that can exist without specific classes.
Instances can have values only for the attached properties. Values must be correctly typed. Range constraints are used for type checking.	Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
Classes encode much of their meaning and behavior through imperative functions and methods.	Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.
Classes can encapsulate their members to private access.	All parts of an OWL/RDF file are public and can be linked to from anywhere else.
Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.	Open world: If there is not enough information to prove a statement true, then it may be true or false.
Role in the Design Process	
Some generic APIs are shared among applications. Few (if any) UML diagrams are shared.	RDF and OWL have been designed from the ground up for the Web. Domain models can be shared online.
Domain models are designed as part of a software architecture.	Domain models are designed to represent knowledge about a domain, and for information integration.
UML, Java, C# etc. are mature technologies supported by many commercial and open-source tools.	The Semantic Web is an emerging technology with some open-source tools and a handful of commercial vendors.
Miscellaneous Features	
Instances are anonymous insofar that they cannot easily be addressed from outside of an executing program.	All named RDF and OWL resources have a unique URI under which they can be referenced.
UML models can be serialized in XMI, which is geared for exchange among tools but not really Web-based. Java objects can be serialized into various XML-based or native intermediate formats.	RDF and OWL objects have a standard serialization based on XML, with unique URIs for each resource inside the file.

pointed out clearly. Whereas general semantic discrepancy between RDF(S) and OWL that is well known in Semantic Web community is addressed here, nicer discussions on non-Unique Name Assumption and Open World Assumption of OWL and their problems with respect to RDF semantics are postponed until Section 6.4 and Chapter 7. Thirdly, CLOS semantics and its computational model are discussed based on the model addressed by Brian Smith, and finally the semantic gap between OWL and object oriented languages addressed by SETF of W3C are pointed out.

All presented in this chapter is theoretical preparations to implement object-oriented language for Semantic Web on top of CLOS. The details of the implementation are described in the next chapter.

Chapter 3

Implementation of RDF, RDFS, and OWL on CLOS

> *"You think you know when you learn, are more sure when you can write, even more when you can teach, but certain when you can program."* (Alan Perlis, Epigrams on Programming (1982))

In this chapter, firstly RDF and RDFS is realized on top of CLOS by straightforward mapping of RDF graph such that a start node of graph to a CLOS object, an edge in graph to a slot-name, and an end node to a slot-value. RDFS class-instance relationship is mapped to CLOS class-instance relationship. The problems arising from such mapping are discussed and solved. Secondly, all of OWL features are realized on top of RDF(S) with preserving RDF(S) semantics. We distinguish substantial sorts and non-substantial sorts, and procedural subsumption computation algorithm for OWL Full is developed. Several OWL specific features are explained with SWCLOS demonstrations.

3.1 Implementation of RDF(S)

3.1.1 Mapping Triples to CLOS Objects

In RDF semantics, a set of triples models a labeled graph called RDF graph. **Fig. 3.1** shows an example of RDF graph[1], and it can be expressed by pure CLOS as follows, using a straightforward

[1]This is taken from the obsolete W3C Working Draft `http://www.w3.org/TR/2002/WD-rdf-schema-20021112/`.

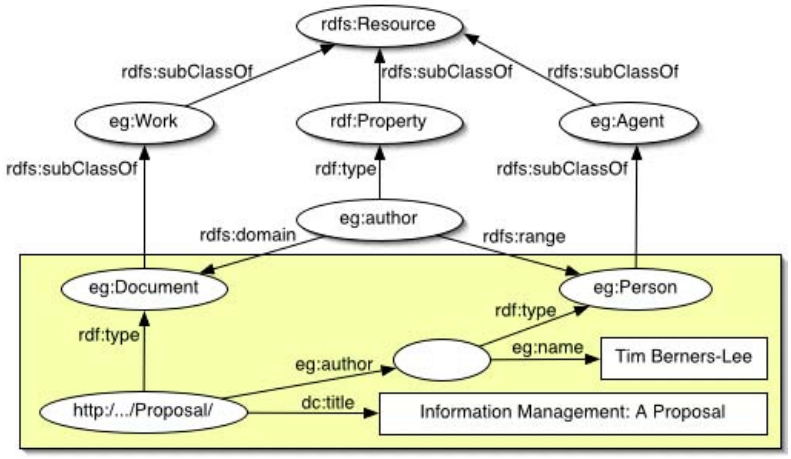

Fig. 3.1: An Example of RDF Graph

mapping such that a triple of *subject/predicate/object* is mapped to CLOS object/slot-name/slot-value.

```
(defpackage rdf
  (:documentation "http://www.w3.org/1999/02/22-rdf-syntax-ns"))
(defpackage rdfs
  (:documentation "http://www.w3.org/2000/01/rdf-schema"))
(defpackage eg
  (:documentation "http://somewhere-for-eg/eg"))
(defpackage dc
  (:documentation "http://dublincore.org/2002/08/13/dces"))

(defclass rdfs::Resource ( ) ((rdf::about :initarg :about)))
(defclass eg::Work (rdfs::Resource) ( ))
(defclass eg::Agent (rdfs::Resource) ( ))
(defclass eg::Person (eg::Agent)
  ((eg::name :initarg :name)))
(defclass eg::Document (eg::Work)
  ((eg::author :initarg :author :type eg::Person)
   (dc::title :initarg :title)))

(defvar eg::Proposal
    (make-instance 'eg::Document
     :author (make-instance 'eg::Person :name "Tim Berners-Lee")
     :title "Information Management: A Proposal"
     :about "http:/···/Proposal/"))
```

In this example, graph nodes are represented by CLOS objects, that is, `rdfs::Resource`, `eg::Work`, etc. for RDF classes are as metaobjects, and `eg::Proposal` and a blank node as instance objects. Graph edges are represented by a slot name, e.g., `eg::author` and `dc::title`. Namely, a triple *subject/predicate/object* in RDF is represented an object, and its pair of slot-name and slot-value.

QName to lisp symbol and namespace to lisp package

Since URI references are inconveniently long, QName is often used to represent elements and attributes in XML documents instead of the URI reference. The appearance of QName is the same as exported lisp symbol[2]. Therefore, a QName in RDF is expressed by a lisp symbol in SWCLOS and a namespace is mapped to a lisp package. At the example demonstration above, packages named `rdf`, `rdfs`, `eg`, and `dc` are defined to implement namespaces with a prefix URI reference in the document option of package, and lisp symbols are used to express QNames. However, note that rdf:type relation is replaced with instance-class relation in CLOS, and rdfs:subClassOf relation is replaced with class-superclass relation in CLOS, because rdf:type and rdfs:subClassOf relations are analogous to the instance-class and class-superclass relations in CLOS, respectively. This issue is discussed more precisely later on.

A resource object for property

However, there are some semantic gaps left by this mapping. CLOS is object-centric but RDF is property-centric. The slot name is not an object in CLOS, but the property in RDF is an instance of rdf:Property. This question is solved by automatically creating a CLOS object of a property resource in RDF universe using RDF entailment rule **rdf1** (See Table 2.4). Namely, a property object is automatically created, the instant SWCLOS finds a slot name undefined as CLOS object. See the following demonstration in SWCLOS.[3]

```
gx-user(2): (addObject rdfs:Resource '((:name yyy)))
#<rdfs:Resource yyy>
gx-user(3): (addObject rdfs:Resource '((:name uuu) (aaa yyy)))
Warning: Entail by rdfs1: aaa rdf:type rdf:Property.
#<rdfs:Resource uuu>
gx-user(4): aaa
#<rdf:Property aaa>
```

[2] A lisp symbol with double colons is internal in a package, and an exported symbol from the package is printed with a single colon.

[3] APIs and their calling sequences of SWCLOS are written at SWCLOS User's Manual [37].

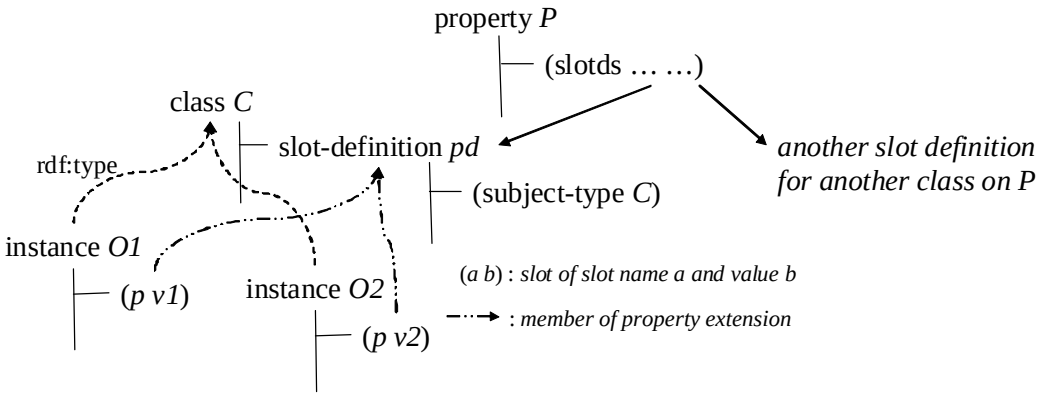

Fig. 3.2: Slot Definition and Slot Extension

Property extension and slot definition

A property as a predicate in triple notation represents a binary relation between two resource objects, or *subject* and *object*. A set of binary relations between subjects and objects on a particular property p is called the *extension* of the property p in RDF, i.e., $EXT^I(p^I)$. On the other hand, whereas slots in CLOS objects are not objects in CLOS semantics but mere placeholders in object memory, a class has slot definitions for its instances. From the viewpoint of RDF, the slot-definition object in CLOS coincides with what represents one of mutually disjoint subsets of a property extension which is partitioned by classes of subjective CLOS objects on a particular property. A slot-definition object is an instance of the `mop:slot-definition` class in CLOS, and it provides the information about slot definitions for instances of the class. See **Fig. 3.2**. Class *C* has a slot-definition *pd* for *C*'s instances, *O*1 and *O*2.

By leveraging this CLOS native slot-definition facility, we created a machinery that collects all elements of the property extension of a particular property. First of all, we newly designed an extra slot option named `subject-type` in the slot-definition object, so that it holds a pointer to the self class of subjective objects on this property. In addition, we designed that all slot definitions for a particular property are booked in `slotds` slot of that property resource. Thus, via slotds value in property *P*, all property extensions of property *P* can be collected, retrieving every pair of subject and object on the slot definition for *P* through the pointer to the subject class of `subject-type` slot. See the following demonstration in SWCLOS.

```
gx-user(7): (collect-all-extensions-of rdfs:comment)
((#<rdfs:Datatype rdf:XMLLiteral> "The class of XML literal values.")
 (#<rdfsClass rdfs:Class> "The class of classes.")
```

```
(#<rdfs:Class rdfs:Resource> "The class resource, everything.")
(#<rdfs:Class rdf:Property> "The class of RDF properties.")
(#<rdfs:Class rdf:Statement> "The class of RDF statements.")
(#<rdfs:Class rdfs:Datatype> "The class of RDF datatypes.")
(#<rdfs:Class rdf:List> "The class of RDF Lists.")
(#<rdfs:Class rdfs:Container> "The class of RDF containers." ...)
```

3.1.2 Type in CLOS and Membership in RDF

Similarity in class transitivity and subsumption between RDFS and CLOS

The role of the class-instance relationship in CLOS is different from that in RDFS. A class in CLOS is a type of instance, and an object of which instances share methods and slot definitions. For instance, a class *C* in CLOS inherits methods and slot definitions from its superclass *D* and other superclasses. The semantics of CLOS is underpinned by the framework of object method inheritance and slot definition inheritance. On the other hand, a class in RDF(S) represents a set to which the instances are classified, and the set is called the *class extension* of the class. The class-superclass relationship in RDFS is the inclusiveness of the class extensions. Namely, the statement that a class *C* is a subclass of *D* means that the class extension of the denotation of *D*, $CEXT^I(D^I)$, includes the class extension of the denotation of *C*, $CEXT^I(C^I)$. See the subsumption rule 2.13 in Chapter 2.

While the semantics of CLOS class-instance is different from that of RDF(S), the class-superclass relationship and class-instance relationship in CLOS work in the same way as RDF(S) with respect to the transitivity of classes and the subsumption. In practice, the RDF subsumption rule **rdfs9**[4](See Table 2.5) and the transitivity rule **rdfs11**[5] (ibid.) are natively realized in CLOS. Therefore, rdfs:subClassOf relation may be straightforwardly mapped to the class-superclass relation in CLOS, and rdf:type relation may be mapped to the class-instance relation. Then, RDFS instances are mapped to CLOS instances, and RDFS classes are mapped to CLOS classes without the violation of the subsumption and transitivity rules in RDF(S). See the following examples in pure CLOS language and compare them with **rdfs9** and **rdfs11**. Where `cl:typep` is a type testing function, and `cl:subtypep` is a class-superclass testing function in Common Lisp; `t` means boolean true, and a form such as `#<standard-class xxx>` is a print form of an object in CLOS.

```
(defclass xxx () ())              -> #<standard-class xxx>
(defclass vvv (xxx) ())           -> #<standard-class vvv>
```

[4] http://www.w3.org/TR/rdf-mt/#rulerdfs9
[5] http://www.w3.org/TR/rdf-mt/#rulerdfs11

```
(defclass uuu (vvv) ())           -> #<standard-class uuu>
(cl:subtypep 'uuu 'xxx)           -> t

(defclass xxx () ())              -> #<standard-class xxx>
(defclass uuu (xxx) ())           -> #<standard-class uuu>
(setq vvv (make-instance 'uuu))   -> #<uuu @ #x2126cc9a>
(cl:typep vvv 'uuu)               -> t
(cl:typep vvv 'xxx)               -> t
```

Metaclasses in RDFS

The straightforward mapping between RDFS class-instance and CLOS class-instance involves that rdfs:Class and rdfs:Datatype in RDFS correspond to the metaclass in CLOS. In the manner of CLOS meta-programming, a class that is a subclass of `cl:standard-class` becomes a metaclass. Therefore, we defined rdfs:Class as a subclass of `cl:standard-class`, then rdfs:Datatype also becomes a CLOS metaclass, because it is a subclass of rdfs:Class.

Realization of rdfs:Class membership loop

As described so far, RDFS classes are realized by CLOS classes, and RDFS metaclasses are realized CLOS metaclasses. However, there was one big obstacle to actually realize them, that is the membership loop of rdfs:Class. As shown in Subsection 2.3.3, the membership loop at cl:standard-class is the key for the realization of reflective object oriented systems. However, Allegro Common Lisp, which is used for realization of SWCLOS, do not permit to set up another membership loop for any other classes. As a result of our effort, a trick for virtually implementing the membership loop for a class is found and utilized within the realms of Allegro Common Lisp. The following demonstration and **Fig. 3.3** shows the essence of this trick.

```
(defclass meta-node (cl:standard-class) ())
(defclass gnode ()                  ; strict class
  () (:metaclass meta-node))
(defclass rdfsClass (meta-node)     ; proxy
  () (:metaclass meta-node))
(defclass rdfs:Class (meta-node)    ; meta
  () (:metaclass rdfsClass))
(defclass rdfsClass (rdfs:Class)
  () (:metaclass meta-node))        ; now twisted
(defclass rdfs:Resource (gnode)
  () (:metaclass rdfs:Class))
(defclass rdfs:Class (meta-node rdfs:Resource) ; final
  () (:metaclass rdfsClass))
```

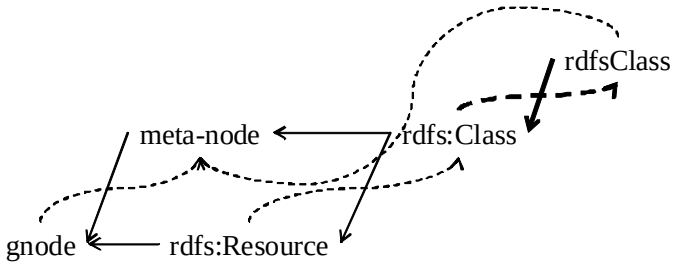

Fig. 3.3: A Trick for Membership Loop for rdfs:Class

Surprisingly the relationship between rdfs:Class and its proxy rdfsClass is the same as the relationship between rdfs:Resource and rdfs:Class in RDF, furthermore cl:standard-object and cl:standard-class, then we particularly named it *twisted* relation.[6] Thereby not only in the semantics of RDF but also in the semantics of CLOS, the membership loop at rdfs:Class is properly implemented. Thus, a method that is defined to rdfs:Class has become applicable to rdfs:Class itself.

```
(class-of rdfs:Class)                              -> #<meta-node rdfsClass>
(cl:subtypep (class-of rdfs:Class) rdfs:Class)  -> t
(cl:typep rdfs:Class rdfs:Class)                   -> t
```

3.1.3 Subsumption of Properties and Domain/Range inheritance

There is a notion of subsumption on property in RDFS, and the domain/range attributes of a property are inherited according to the super-sub relation of properties. However, CLOS is not equipped with subsumption facility among instances[7], as well as ordinary OOPLs are not. Therefore, we realized subsumption notion of the property in RDFS. Furthermore, we put the domain/range constraint and their inheritance in the property subsumption. See the following examples in SWCLOS on the domain constraint, in which **rdfs5**[8], **rdfs7**[9], and **rdfs2**[10] are expressed here.

```
(defProperty aaa (rdfs:subPropertyOf bbb)) -> #<rdf:Property aaa>
(defProperty bbb (rdfs:domain xxx))        -> #<rdf:Property bbb>
(defIndividual uuu (aaa yyy))              -> #<xxx uuu>
(cl:typep uuu xxx)                         -> t
```

To embody the range constraint, the range information that exists in a property object as a resource is transferred to the type option in the relevant slot-definition object. The system checks the range

[6] This issue is described more in detail in Chapter 6.
[7] Properties are instances of rdfs:Property.
[8] http://www.w3.org/TR/rdf-mt/#rulerdfs5
[9] http://www.w3.org/TR/rdf-mt/#rulerdfs7
[10] http://www.w3.org/TR/rdf-mt/#rulerdfs2

constraint for objects in the slot-definition at the class when an instance of class is created. The following shows the realization of **rdfs5**, **rdfs7**, and **rdfs3**[11].

```
(defProperty aaa (rdfs:subPropertyOf bbb)) -> #<rdf:Property aaa>
(defProperty bbb (rdfs:range zzz))         -> #<rdf:Property bbb>
(defIndividual uuu (aaa yyy))              -> #<xxx uuu>
(cl:typep yyy zzz)                         -> t
```

3.1.4 Tailored Slot Specification

A class in CLOS is also an object called *metaobject*. A metaobject for class is equipped with a special slot for slot-definitions that specify slot structures of its own instances. Furthermore, the slot definition *per se* is also a CLOS object (*slot-definition-object*) that is equipped with slots for slot definition data. A slot definition datum in a slot of the slot definition object is called *slot option*. As described at Subsection 2.3.1, in the instance creation process, all slot definitions of the direct class of an instance and its superclasses are collected, then the collection turns out an instance of `mop:standard-effective-slot-definition` and it is stored in a slot named `mop:slots` in the class to which the slot is defined.

To distinguish slot definitions for RDF/OWL objects in RDF universe and OWL universe from CLOS slot definitions for standard CLOS objects, we newly defined `Property-direct-slot-definition` and `OwlProperty-direct-slot-definition` as a subclass of `mop:standard-direct-slot-definition`, as well as `Property-effective-slot-definition` and `OwlProperty-effective-slot-definition` as a subclass of `mop:standard-effective-slot-definition`, and then we tailored the definition of slots. **Fig. 3.4** depicts the new slot definitions that are defined for RDF and OWL.

We designed that the class constraint by rdfs:range and owl:onProperty for slot value is stored into the CLOS native type option of slot definition, then we additionally set extra two options for cardinality constraint `maxcardinality` and `mincardinality` in the instance of `OwlProperty-direct-slot-definition`. In the MOP process for making effect slot definition objects at instance creation, the most specific class are stored into the type option, Therefore, we adapted this facility to compute inherited rdfs:range and owl:onProperty type constraints for the slot value, and minimum value among owl:maxCardinality constraints and the maximum value among owl:minCardinality constraints are also computed and stored into `maxcardinality` option and `mincardinality`, respectively.

[11] `http://www.w3.org/TR/rdf-mt/#rulerdfs3`

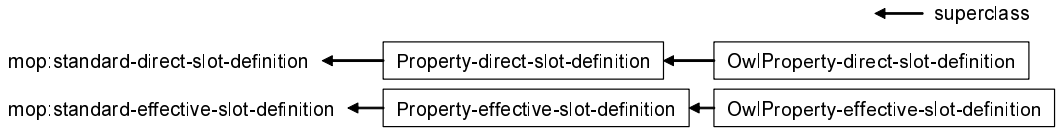

Fig. 3.4: Slot Definitions Dedicated to RDF and OWL

The slot option `subject-type` is used for book-keeping the class of triple subjects to which this slot is allocated.

3.1.5 Slot Definition On-Demand from Instance Objects

CLOS is a class-based system. Namely, a class must be defined before its instance creation. A slot definition is needed at a class before setting slot value to an instance of a class. However, from the viewpoint of graph based model like RDF, it is desirable that it is able to add a slot with slot value into an object anytime and anywhere. So, we implemented the machinery of such slot adding functionality. A slot definition is automatically generated and attached to a class on demand, if it is required at an instance of the class and when the class has no corresponding slot definitions. See the following demonstration.

```
gx-user(3): (defResource uuu (rdf:type rdfs:Class))
#<rdfs:Class uuu>
gx-user(4): (defResource aaa (rdf:type uuu)
              (hasSomething "Wonderful!"))
Warning: Entail by rdf1: hasSomething rdf:type rdf:Property.
#<uuu aaa>
gx-user(5): (slot-value aaa 'hasSomething)
"Wonderful!"
```

In this demonstration, a slot name `hasSomething` was given to an instance of `uuu` without its slot definition on `uuu`. The system automatically created the definition of slot named `hasSomething` on `uuu`, then the instance named `aaa` was made. Note that slot definitions are also defined when rdfs:domain is defined to a property, and when owl:onProperty restriction is defined to a class and a property.

3.1.6 Single Class in CLOS and Multiple Classes in RDF(S)

In the RDF graph model, an object may be typed to more than one class through rdf:type property. However, a CLOS object cannot belong to multiple classes. A CLOS class is a prototype to create

its instances; therefore instances must inevitably belong to a single class. To solve this problem, we introduced invisible classes *shadowed-class* that may be a subclass of visible multiple classes from the RDF viewpoint. For example, a vintage wine vin:SaucelitoCanyonZinfandel1998, which is classified to vin:Vintage and vin:Zinfandel in Wine Ontology[12], is an instance of vin:Zinfandel.2 in CLOS, which is invisible in OWL and a subclass of vin:Vintage and vin:Zinfandel. When an object belongs to multiple classes, SWCLOS sets up a *shadowed-class* that has multiple classes in its superclass list, and the object is classified to the shadowed-class. See the following demonstration.

```
gx-user(2): (defResource ItalianWine (rdf:type rdfs:Class))
#<rdfs:Class ItalianWine>
gx-user(3): (defResource RedWine (rdf:type rdfs:Class))
#<rdfs:Class RedWine>
gx-user(4): (defResource MyFavoriteWine (rdf:type ItalianWine)
              (rdf:type RedWine))
Warning: Multiple classing with (#<rdfs:Class ItalianWine> #<rdfs:Class RedWine>)
         for #<ItalianWine MyFavoriteWine>
#<ItalianWine.0 MyFavoriteWine>
gx-user(5): (typep MyFavoriteWine ItalianWine)
t
t
gx-user(6): (typep MyFavoriteWine RedWine)
t
t
```

3.1.7 Forward Reference and Proactive Entailment

The forward reference was enabled by means of proactive entailments in which undefined resources are defined without human intervention using entailment rules. In CLOS original functionality, CLOS creates an undefined but referred class as a class under `mop:forward-referenced-class` in order to enable the forward reference. However, an attempt to make an instance of a forward reference class causes an alarm in CLOS. The forward referenced class in CLOS must be defined by the time of its instance creation. This function is insufficient for RDF forward reference. Fortunately, there are a number of RDF (Table 2.1 though 2.4) and RDFS rules (Table 2.5) and OWL entailment rules (Table 2.6) in addition to the monotonicity principle in Semantic Web. Therefore, if we encounter an undefined class reference in reading an RDF/OWL file, we can create the undefined class as the most specific concept in the context by applying various RDF(S) and OWL entailment rules for the context without the contradiction in definitions that will appear later on. For instance, rule

[12] `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`

rdf1[13] can be utilized for an undefined predicate to be created as an instance of rdf:Property, and rule **rdfs4**[14] assures for a subject and an object in triple to be defined as an instance of rdfs:Resource. Needles to say, domain constraints, range constraints, and onProperty/owl:allValuesFrom restrictions are also available to such forward reference entailing. Such entailments in run time are enabled by lisp dynamic programming characteristics represented by changing class and reinitializing object facilities of CLOS.

The proper definition may be used to refine forward referenced classes and objects more precisely. For instance, a class in $\boldsymbol{C}^{\mathcal{I}}$ is changed in $\boldsymbol{OC}^{\mathcal{I}}$, if it is found that the class has a slot of owl:unionOf or owl:intersectionOf.

The following demonstrates that undefined three resources in a statement of triple, one property and two resources are actually defined using entailment rules **rdf1**, **rdfs4a**, and **rdfs4b**.

```
gx-user(2): (defIndividual uuu (aaa yyy))
Warning: Entail by rdf1: aaa rdf:type rdf:Property.
#<|rdfs:Resource| uuu>
gx-user(3): aaa
#<rdf:Property aaa>
gx-user(4): yyy
#<|rdfs:Resource| yyy>
```

Where `|rdfs:Resource|` is an alternative to rdfs:Resource, whereby a proper rdfs:Resource in CLOS are treated as an abstract class that do not directly hold any instances.

3.2 RDF(S) Demonstration in SWCLOS

The semantics of domain constraint (2.17) and range constraint (2.18) is demonstrated below.[15]

```
gx-user(2): (defProperty hasColor
        (rdfs:domain Wine)
        (rdfs:range Color))
Warning: Range entailX3 by rdfs:domain: Wine rdf:type rdfs:Class.
Warning: Range entailX3 by rdfs:range: Color rdf:type rdfs:Class.
#<rdf:Property hasColor>
gx-user(3): (defIndividual ElyseZinfandel
        (hasColor Red))
Warning: Range entailX3 by hasColor: Red rdf:type Color.
#<Wine ElyseZinfandel>
gx-user(4): Red
#<Color Red>
```

[13] http://www.w3.org/TR/rdf-mt/#rulerdf1

[14] http://www.w3.org/TR/rdf-mt/#rulerdfs4

[15] BNF syntax is described in SWCLOS User's Manual [37].

In this example, `Wine` and `Color` are defined as class (an instance of rdfs:Class) using range entailment in the form of `hasColor` definition. Then, `ElyseZinfandel` is defined as an instance of `Wine` because of the domain constraint of `hasColor` with the range constraint that entailed `Red` is an instance of `Color`.

3.3 OWL Full on Top of RDF(S)

As mentioned at Section 2.1 and 2.2, OWL DL is not compatible to RDF. This section describes how OWL is realized on top of RDF(S).

3.3.1 RDF Compatibility of OWL

RDF compatibility of OWL is discussed at OWL Semantics Documentation Chapter 5 [58]. In this documentation, it is clearly stated in the table titled "Conditions concerning the parts of OWL universe and syntactic categories" that i) the class extension of owl:Thing is included in the class extension of rdfs:Resource, ii) the class extension of owl:Class is included in the class extension of rdfs:Class, and iii) four kinds property extensions are included in the property extension of rdf:Property. Namely, the statements are formalized as follows. See also Section 2.2.

$$\boldsymbol{OT}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Thing^{\mathcal{I}}) \subset \boldsymbol{R}^{\mathcal{I}} = CEXT^{\mathcal{I}}(rdfs{:}Resource^{\mathcal{I}}) \quad (3.1)$$

$$\boldsymbol{OC}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Class^{\mathcal{I}}) \subset \boldsymbol{C}^{\mathcal{I}} = CEXT^{\mathcal{I}}(rdfs{:}Class^{\mathcal{I}}) \quad (3.2)$$

$$\boldsymbol{OP}_I^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}ObjectProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} = CEXT^{\mathcal{I}}(rdf{:}Property^{\mathcal{I}}) \quad (3.3)$$

$$\boldsymbol{OP}_D^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}DatatypeProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \quad (3.4)$$

$$\boldsymbol{OP}_A^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}AnnotationProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \quad (3.5)$$

$$\boldsymbol{OP}_O^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}OntologyProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \quad (3.6)$$

However, the document continues to state the following statement;

> "There are two different styles of using OWL. In the more free-wheeling style, called OWL Full, the three parts of OWL universe are identified with their RDF counterparts, namely the class extensions of rdfs:Resource, rdfs:Class, and rdf:Property. In OWL Full, as in RDF, elements of OWL universe can be both an individual and a class, or, in fact, even an individual, a class, and a property. In the more restrictive style, called OWL DL here, the three parts are different from their RDF counterparts and, moreover,

> pairwise disjoint. The more-restrictive OWL DL style gives up some expressive power in return for decidability of entailment. Both styles of OWL provide entailments that are missing in a naive translation of the DAML+OIL model-theoretic semantics into the RDF semantics."(OWL Semantics Chapter 5, [58])

Namely, it follows axioms in OWL Full;

$$\boldsymbol{OT}^{\mathcal{I}} = \boldsymbol{R}^{\mathcal{I}} \tag{3.7}$$

$$\boldsymbol{OC}^{\mathcal{I}} = \boldsymbol{C}^{\mathcal{I}} \tag{3.8}$$

$$\boldsymbol{OP}_I^{\mathcal{I}} \cup \boldsymbol{OP}_D^{\mathcal{I}} \cup \boldsymbol{OP}_A^{\mathcal{I}} \cup \boldsymbol{OP}_O^{\mathcal{I}} = \boldsymbol{P}^{\mathcal{I}}. \tag{3.9}$$

Whereas it follows axioms in OWL DL;

$$\boldsymbol{OT}^{\mathcal{I}} \not\subseteq \boldsymbol{R}^{\mathcal{I}} \tag{3.10}$$

$$\boldsymbol{OC}^{\mathcal{I}} \not\subseteq \boldsymbol{C}^{\mathcal{I}} \tag{3.11}$$

$$\boldsymbol{OP}_I^{\mathcal{I}} \cup \boldsymbol{OP}_D^{\mathcal{I}} \cup \boldsymbol{OP}_A^{\mathcal{I}} \cup \boldsymbol{OP}_O^{\mathcal{I}} \not\subseteq \boldsymbol{P}^{\mathcal{I}} \tag{3.12}$$

$$\boldsymbol{OT}^{\mathcal{I}} \cap \boldsymbol{OC}^{\mathcal{I}} = \emptyset \tag{3.13}$$

$$\boldsymbol{OT}^{\mathcal{I}} \cap (\boldsymbol{OP}_I^{\mathcal{I}} \cup \boldsymbol{OP}_D^{\mathcal{I}} \cup \boldsymbol{OP}_A^{\mathcal{I}} \cup \boldsymbol{OP}_O^{\mathcal{I}}) = \emptyset \tag{3.14}$$

$$\boldsymbol{OC}^{\mathcal{I}} \cap (\boldsymbol{OP}_I^{\mathcal{I}} \cup \boldsymbol{OP}_D^{\mathcal{I}} \cup \boldsymbol{OP}_A^{\mathcal{I}} \cup \boldsymbol{OP}_O^{\mathcal{I}}) = \emptyset. \tag{3.15}$$

However, we adopted the third way in order to realize OWL Full on top of RDF. Namely, as described in OWL Semantics Chapter 5, [58] and Section 2.2, OWL universe is included in RDF universe. See above equations, (3.1), (3.2), and (3.3) through (3.6). OWL semantics is superimposed to entities in OWL universe that is ruled by RDF semantics.

In fact, it is possible to realize the following axioms just by means of reading the OWL definition file [16] in the RDF semantics and the RDF universe using RDFS module in SWCLOS.

[16] `http://www.w3.org/2002/07/owl.rdf`

$$owl{:}Class^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{3.16}$$

$$\boldsymbol{OC}^{\mathcal{I}} \subset \boldsymbol{C}^{\mathcal{I}} \tag{3.17}$$

$$owl{:}Restriction^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \tag{3.18}$$

$$\boldsymbol{OR}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Restriction^{\mathcal{I}}) \subset \boldsymbol{C}^{\mathcal{I}} \tag{3.19}$$

$$owl{:}Thing^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \tag{3.20}$$

$$\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}} \tag{3.21}$$

$$owl{:}ObjectProperty^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{3.22}$$

$$\boldsymbol{OP}_I^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}ObjectProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \tag{3.23}$$

$$owl{:}DatatypeProperty^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{3.24}$$

$$\boldsymbol{OP}_D^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}DatatypeProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \tag{3.25}$$

$$owl{:}AnnotationProperty^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{3.26}$$

$$\boldsymbol{OP}_A^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}AnnotationProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \tag{3.27}$$

$$owl{:}OntologyProperty^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{3.28}$$

$$\boldsymbol{OP}_O^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}OntologyProperty^{\mathcal{I}}) \subset \boldsymbol{P}^{\mathcal{I}} \tag{3.29}$$

Note that $\boldsymbol{OT}^{\mathcal{I}} \subset \boldsymbol{R}^{\mathcal{I}}$ (3.1) or (3.21) is entailed by RDF entailment rule **rdfs4a**[17].

Thus, SWCLOS is able to distinguish and control both resources that are ruled by RDF semantics and objects that are ruled by OWL semantics[18], whereas OWL Full style by W3C cannot distinguish $\boldsymbol{OC}^{\mathcal{I}}$ against $\boldsymbol{C}^{\mathcal{I}}$, $\boldsymbol{OT}^{\mathcal{I}}$ against $\boldsymbol{R}^{\mathcal{I}}$.

In addition, we added the followings axiom in order to make OWL classes belong to OWL universe, because such axiom is not defined in the OWL definition file. [19]

Axiom 1.

$$\boldsymbol{OC}^{\mathcal{I}} \subset \boldsymbol{OT}^{\mathcal{I}} \tag{3.30}$$

Namely, the class extension of the denotation of owl:Class URI reference is a subset of the class extension of the denotation of owl:Thing URI reference. Note that this is the same as the relationship

[17] `http://www.w3.org/TR/rdf-mt/#rulerdfs4`

[18] Actually, every RDF resources is an instance of rdfs:Resource, and every OWL object is an instance of owl:Thing.

[19] Note that the document [58] actually describes (3.30).

between rdfs:Class and rdfs:Resource. Thus, this axiom makes vin:Zinfandel and vin:Wine belong to OWL universe as well as their individuals in Wine Ontology.

This axiom is written as follows in SWCLOS S-expression.

```
(defResource owl:Class (rdfs:subClassOf owl:Thing))
```

From the object oriented view, this axiom implies that every class in OWL universe inherits properties defined at owl:Thing for individuals. Thus, classes in OWL universe are enabled to have owl:sameAs and owl:differentFrom properties, then enabled to be treated classes as individuals.

3.3.2 Anonymous Restriction Classes for Properties

While OWL object-centric expressions look like object expressions in OOPLs rather than RDF graphs, they still obey RDF syntax and semantics in OWL Full. Therefore, the property restrictions in OWL become anonymous classes as instances of owl:Restriction[20]. Thus, the subjective CLOS object in the expression is defined as a subclass of the anonymous restriction classes that appears within rdfs:subClassOf or owl:intersectionOf representations. For example, in the definition of vin:Wine in the Wine Ontology, vin:Wine has two anonymous classes, the restriction for the cardinality and the value restriction of vin:Winery, on the vin:hasMaker property.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
          ...
</owl:Class>
```

[20] As mentioned earlier, every class in CLOS is an object called metaobject. Furthermore, class metaobject may have no name in CLOS. See also the demonstration in Section 6.2

As described in Subsection 2.3.1, multiple pieces of information that exist among superclasses upon a particular property with different values for the same restriction are collected and the most special concepts or the strictest constraints are computed for the value restriction or existential restriction. Thus, those facilities enabled the satisfiability-checking for the slot-value against constraints in the instance creation. For instance, after loading the Wine Ontology and the Food Ontology, an attempt at creating an instance of the unsatisifiable concept `TheSpecialCourse` causes a satisfiability error.

```
(defResource TheSpecialCourse (rdf:type owl:Class)
  (owl:intersectionOf
    food:RedMeatCourse
    (owl:Restriction (owl:onProperty food:hasFood)
                     (owl:allValuesFrom food:Fruit))))

(defIndividual No1SpecialCourse (rdf:type TheSpecialCourse)
  (food:hasFood food:Meat food:Bananas))

Error: Unsatisfiable by disjoint pair in
       (#<owl:Class food:Fruit> #<owl:Class food:RedMeat>) for
       TheSpecialCourse food:hasFood
```

Here, the value restriction `food:RedMeat` is inherited from `food:RedMeatCourse` and the value restriction `food:Fruit` is defined at the concept `TheSpecialCourse`. The satisfiability error happened at the creation of `No1SpecialCourse`, an instance of `TheSpecialCourse`.

The property constraints in OWL on owl:onProperty generate anonymous classes as instances of owl:Restriction (owl:Restriction is a metaclass), a restriction that is attached to a class in the description of rdfs:subClassOf or owl:intersectionOf turns out in CLOS a superclass of the class to which the restriction is attached, then the restriction value that is transferred into the type option in the slot definition object of the anonymous restriction class is inherited by the class that the restriction is attached and all of its subclasses. Thus, CLOS inheritance machinery is reasonable from the viewpoint of the property restriction in OWL semantics.

3.3.3 Axiomatic Complete Relations

Among the many OWL properties, only four, i.e., owl:intersectionOf, owl:unionOf, owl:complementOf, and owl:oneOf make axiomatic complete assertions without using the owl:equivalentClass.[21] In

[21] `http://www.w3.org/TR/owl-ref/#DescriptionAxiom`

other words, these properties define complete equivalency upon the binary relation of concepts. For example, the following asserts the definition of WhiteBordeaux. If something is known as Bordeaux and WhiteWine, it is concluded to be a WhiteBordeaux.

```
<owl:Class rdf:ID="WhiteBordeaux">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Bordeaux" />
    <owl:Class rdf:about="#WhiteWine" />
  </owl:intersectionOf>
</owl:Class>
```

Similarly, the following assertion defines WineColor, which has the enumerative membership of White, Rose, and Red, so that the instance of WineColor is exactly one of the three, and not any of others.

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White" />
    <owl:Thing rdf:about="#Rose" />
    <owl:Thing rdf:about="#Red" />
  </owl:oneOf>
</owl:Class>
```

Therefore, it is not necessary to place the open world assumption upon such axiomatic complete relational properties. If we find the right-hand side (body of object-centered expression) of such an equation (object-centered expression) matches the database, then we may conclude the left-hand side (subjective term) without worry about other statements. Conversely, we must assert consistent knowledge of a subject in one assertion with respect to these four properties.

Suppose the following example for owl:intersectionOf. The system concludes that QueenElizabethII should be a woman, because it is asserted that a person who has gender female is a woman, and it is also asserted that QueenElizabethII is an instance of Person and hasGender Female. Here note that the system proactively made the entailment without demand or query from users.

```
(defIndividual Female (rdf:type Gender) (owl:differentFrom Male))
 -> #<Gender Female>
(defResource Person (rdf:type owl:Class)
  (owl:intersectionOf
    Human
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:cardinality 1))))
```

```
 -> #<owl:Class Person>
(defResource Woman (rdf:type owl:Class)
  (owl:intersectionOf
    Person
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:hasValue Female))))
 -> #<owl:Class Woman>
(defIndividual QueenElizabethII (rdf:type Person)
  (hasGender Female))
 -> #<Woman QueenElizabethII>
```

3.3.4 Substantial Properties and Non-Substantial Properties

OWL has many properties that rule the inclusiveness of concepts, i.e., rdfs:subClassOf, owl:intersectionOf, owl:unionOf, owl:equivalentProperty, etc. From the viewpoint of DL, they have same strength for subsumption decidability. However, from the viewpoint of ontology engineering and software engineering, we must discriminate substantial ones and non-substantial ones for ruling subsumption. Borgida [6] argued that one should deal with individual objects that remain related rather than volatile references. Mizoguchi [52, 53] claimed that the IS-A relation (substantial sort) should comply with single inheritance from the viewpoint of Ontology Engineering, whereas an object may have multiple roles (non-substantial sort). Kaneiwa and Mizoguchi [29] developed a formal ontology for property classification and extended Order-Sorted Logic to the property classification.

It is also important for the ontology and database maintainability to distinguish persistent relations and temporal relations. In SWCLOS, the rdfs:subClassOf relation in the RDF universe is mapped onto the class-superclass relation in CLOS, and a CLOS object as an rdfs:subClassOf property value is placed in the `direct-superclasses-list` slot of the subjective class object. However, when a property `p1` in OWL universe is a subproperty of or equivalent property to rdfs:subClassOf, should we place `p1`'s value into the direct-superclasses slot in the class? Note that a superclass in CLOS affects the slot structure of its instances. In other words, what property in OWL may affect the structural variation in CLOS due to the slot inheritance, and what property must not affect the structural variation in CLOS? We specified that rdfs:subClass, owl:intersectionOf, and owl:unionOf should affect the variation, but owl:equivalentClass, owl:equivalentProperty and other properties, including subproperties and equivalent properties of rdfs:subClass, owl:intersectionOf, or owl:unionOf, should affect subsumption reasoning but not the structural variation.

Conversely, we should define the substantial and persistent subsumption with rdfs:subClassOf, owl:intersectionOf, and owl:unionOf, and the non-substantial subsumption should be defined through other properties. The substantial subsumption may cause proactive entailment, but the non-substantial

subsumption should not cause any structural variation in the entailment. Thus, such discrimination of substantial and non-substantial subsumptions allows us to add and delete relations and makes it easy to maintain ontologies.

3.3.5 Extended Structural Subsumption Algorithm

We extended the *structural subsumption algorithm* in the $\mathcal{FL}_0$ level [5], which is applicable only to conjunction (owl:intersectionOf) of concepts and value restriction (owl:allValuesFrom), so as to include disjunction (owl:unionOf), disjointness (owl:disjointWith), negation (owl:complementOf) of concepts; equivalency (owl:sameAs, owl:equivalentClass, owl:equivalentProperty); symmetric relation (owl:SymmetricProperty), functional relation (owl:InverseFunctionalProperty), inverse-functional relation (owl:FunctionalProperty); full existential restriction (owl:someValuesFrom), filler restriction (owl:hasValue), and cardinality restriction (owl:maxCardinality, owl:minCardinality, owl:cardinality). This original structural subsumption is described as follows [5].

Let B_i and A_j be distinct names of atomic or complex concepts, R_j and S_i be distinct names of roles, C_j and D_i be $\mathcal{FL}_0$-concept descriptions, and let

$$A_1 \sqcap \cdots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \cdots \sqcap \forall R_n.C_n$$

be the normal form of the $\mathcal{FL}_0$-concept description of C, and let

$$B_1 \sqcap \cdots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \cdots \sqcap \forall S_l.D_l$$

be the normal form of the $\mathcal{FL}_0$-concept description of D. Here, $\sqcap$ represents the intersection of concepts. Then $C \sqsubseteq D$ (inclusiveness of concepts) holds iff the following two conditions hold:
(1) For all i in $1 \leq i \leq k$, there exists j in $1 \leq j \leq m$ such that $A_j \sqsubseteq B_i$.
(2) For all i in $1 \leq i \leq l$, there exists j in $1 \leq j \leq n$ such that $S_i = R_j$ and $C_j \sqsubseteq D_i$.

The computation of the class-superclass relation in CLOS conveys the semantics of weak subsumption in RDF universe. However, owl:intersectionOf and owl:unionOf relation show the strong subsumption in OWL universe, and owl:equivalentWith and owl:oneOf relation also have iff relation. To compute the strong subsumption in OWL universe, function `subsumed-p` computes the inclusiveness of the class extensions between two concepts according to the OWL semantics using ternary truth value. Note that direct relation of owl:intersectionOf and owl:unionOf are computed firstly in the RDF graph links or CLOS, because the direct relations of them are replaced into super/subclass relation in CLOS. However, `subsumed-p` secondly computes the strong subsumption

of class extensions, even if two classes are not directly connected. The weak subsumption in the substantial properties (rdfs:subClassOf, owl:intersectionOf, owl:unionOf) is computed through the CLOS class-superclass relationship, and the non-substantial subsumption is calculated with this extended structural subsumption algorithm.

In this algorithm, the top concept $\top$ ($owl{:}Thing^{\mathcal{I}}$) substantially subsumes every concept of $CEXT^{\mathcal{I}}(owl{:}Class^{\mathcal{I}})$ in the CLOS class-subclass relation, but the bottom concept $\bot$ ($owl{:}Nothing^{\mathcal{I}}$) is virtually subsumed by other concepts through this extended structural subsumption algorithm. For classes in OWL universe, the following algorithm returns one of true/false/unknown as truth value.

Where symbol $\equiv$ is used for the equivalency of Lisp objects (equal[22]). First, the top and the bottom are checked. Next, all equivalent classes on C and D is retrieved and checked. The subsumption of classes for owl:oneOf is firstly checked before ordinary computation. $\preceq$ designates the subsumption in CLOS, then it checks the direct relation of owl:intersectionOf and owl:unionOf in additon to rdfs:subClassOf. Then, indirect subsumption in owl:intersectionOf and owl:unionOf is checked. $\simeq$ means class equivalency considering owl:FunctionalProperty and owl:InverseFunctionalProperty, and $\asymp$ designates the relationship of complement and disjoint.

on $C, D \in CEXT^{\mathcal{I}}(owl{:}Class^{\mathcal{I}})$
  if $C \equiv D$ **then return** true
  if $D \equiv \top$ **then return** true
  if $C \equiv \bot$ **then return** true
  if $D \equiv \bot$ **then return** false
  if $C \equiv \top$ **then return** false
  for some $\bar{C}$ **in** equivalent-classes-of(C)
    for some $\bar{D}$ **in** equivalent-classes-of(D)
      if oneOf($\bar{C}$) **and** oneOf($\bar{D}$) **then**
        if oneOf($\bar{C}$)$\subseteq$oneOf($\bar{D}$) **then**
          return true **else return** false
      else if $\bar{C} \preceq \bar{D}$ **then return** true
      else if $\bar{D} \preceq \bar{C}$ **then return** false
      else if intersect-subsumed($\bar{C}$,$\bar{D}$) **then return** true
      else if union-subsumed($\bar{C}$,$\bar{D}$) **then return** true
      else if $\bar{C} \simeq \bar{D}$ **then return** true
      else if $\bar{C} \asymp \bar{D}$ **then return** false
      else return unknown

[22]It is for string equivalency in Lisp.

As shown in Subsection 3.3.3, on the relation of owl:intersectionOf and owl:unionOf, two classes of which the components (the right hand sides) are equivalent are equivalent (in the left hand sides). The subsumption computation algorithm for owl:intersectionOf is described as follows.

$DD \leftarrow$ unfold(intersects-of($\bar{D}$))
for every $D_{cls} \in$ collect-concepts-in(DD)
  if for some $C_{cls} \in$ all-superconcepts-of($\bar{C}$)
    $C_{cls} \sqsubseteq D_{cls}$ **then**
    if restrictions-of(DD)= $\emptyset$ **then return** true
    else if for every $R \in$ roles(restrictions-of(DD))
      $cslot \leftarrow$ slot-definition(C_{cls},R)
      $modelsList \leftarrow$ generate-models($cslot$)
      for some $models$ **in** $modelsList$
        if for every $model$ **in** $models$
          satisfy($model$,restrictions-of(DD,R))
        then return true
    else return false

DD is a set of unfolded classes for $\bar{D}$ on owl:intersectionOf. So, all components in DD are atomic. D_{cls} is a part of non property restriction in DD. C_{cls} is a collection of non property restriction from all superclasses of C. In this computation, firstly the subsumption for concepts, secondly property constraints are performed. Here, symbol $\sqsubseteq$ stands for inclusiveness of class extensions (namely checking by `subsumed-p` itself). In case of property constraints, satisfiability is checked by seeing models. Here, *cslot* is slot definitions on property R for C_{cls}. As mentioned so far, this slot definition holds all inherited cardinality and type restrictions. According to these restriction, all possible models are generated. Firstly, plausible models are generated based on existence of individuals by owl:hasValue and owl:someValuesFrom. By owl:hasValues the value can be one of interpretation models, and by owl:someValuesFrom the instance of constraint can be one of models. Note that instance of owl:someValuesFrom is a variable over the owl:someValuesFrom value class. In case of max cardinality restriction given, the possible models are generated up to the numbers of cardinality restriction, after that, the next existent constraint is superimposed onto one of models so as to map all existing models. Thus, generally, a set of possible sets of models are obtained. The restriction of owl:allValuesFrom is applied onto every possible models. Then, the number of models may be reduced by unsatisfiability. In a very simple case, say, max cardinality is one, the value of owl:hasValue is v, type A for owl:someValuesFrom, type B for owl:allValuesFrom, then v becomes an instance of the most specific concepts of A, B, and its original class, and the satisfiable model is

one. At the last, the generated models are checked against D whether or not it satisfies D's property constraints.

Obviously, this algorithm includes self-recursion. However, this computation is terminated, because CLOS does not accept direct and indirect loop in super-subclass relationship, and the class precedence list for $\bar{C}$ and unfolding classes for $\bar{D}$ prevents the occurrence of loop in tracing rdfs:subClassOf and owl:unionOf definition (e.g., B is a subclass of C and C is a union of A and B)[23].

In SWCLOS, all individuals (including classes as individual) can be checked for subsumption with respect to owl:TransitiveProperty and owl:sameAs. See the followings.

on C, D **in** $CEXT^I$ ($owl{:}Thing^I$)
  if $C \doteq D$ **then return** true
  for some $prop$ **in** intersection(all-transitive-props-of(C), all-transitive-props-of(D))
    for some D_x **in** same-things-of(D)
      for some C_x **in** same-things-of(C)
        if transitively-sub-on($prop$, D_x , D_x) **then return** true
        else return false

Where symbol $\doteq$ means the equality as OWL individual (owl:sameAs).

3.3.6 Satisfiability Check

Proactive entailment reduces the load of the satisfiability check. For example, when users attempt to define an object ambiguously (to define an object in a more abstract class), if the domain and range constraints are available, the system defines an object more specifically (defines an object in a more special class), by fitting the domain and range restriction. Nevertheless, the satisfiability check is useful to prevent ones from importing bugs into ontologies. We implemented satisfiability checking on the domain and range, value restriction, filler restriction, cardinality, disjoint-pair, etc., and Table 3.1 summarized additional unsatisfiability rules in the OWL definition for SWCLOS.

3.3.7 OWL Entailment Rules

The total number of OWL entailment rules is not known yet. In addition to the entailment rules which ter Horst disclosed [72](See Table 2.6), SWCLOS added five axioms shown in Table 3.2, and 18 entailment rules shown in Table 3.3.

[23]Note that SWCLOS does not accept cyclic loop in RDF graph for other properties.

Table 3.1: Unsatisfiability in OWL added to SWCLOS

Unsatisfiability	Conditions
unsatisfiability1	C `oneOf` { $x_i \ldots$ }
	y `type` C
	y `differentFrom some` x_i
unsatisfiability2	x `differentFrom` y
	x `sameAs` y
unsatisfiability3	C `disjointWith` D
	D `equivalentClass` C
unsatisfiability4	C `disjointWith` D
	x `type` C
	x `type` D

Table 3.2: Additional OWL Axioms for SWCLOS

axiom1	`owl:Thing rdfs:subClassOf rdfs:Resource .`
axiom2	`owl:Class rdfs:subClassOf owl:Thing .`
axiom3	`owl:FunctionalProperty rdf:type owl:Class .`
axiom4	`owl:InverseFunctionalProperty rdf:type owl:Class .`
axiom5	`owl:FunctionalProperty owl:disjointWith owl:InverseFunctionalProperty .`

In this subsection and succeeding sections, **rdf*** and **rdfs**** means one of RDF entailment rules and RDFS entailment rules, respectively. A description of **rdfp**** means one of entailment rules by ter Horst. If **rule*** found, it means one of entailement rules in Table 3.3.

Since SWCLOS is a procedural reasoning system, all of entailment rules are implemented into SWCLOS programming code. Therefore, we might miss to find out a proper position where premises of rules match context and to insert entailment procedures. However, note that the Tableau Algorithms are insufficient for implementing proactive entailment. The work of Tableau is to test the membership of individuals and the subsumption relation among classes by means of refutation. The prover based on refutation does work with given refutation. So, to implement proactive entailments, we must find out refutation when and where we think entailments are required. In order to perform proactive entailments, we need to sense the situation that matches the premise of entailment rules and in which an entailment is deductive.

Hereafter, we introduce many entailment rules in OWL and discuss how these rules are implemented in our system.

SameAs Group, EquivalentClass Group, EquivalentProperty Group

The owl:sameAs relation is reflexive (**rdfp6**) and transitive (**rdfp7**). Thus, all related individuals make one group upon owl:sameAs. The group information that is a collection of related individuals in owl:samaAs is registered to each individual of the group. The owl:equivalentClass owl:equivalentProperty is also reflexive (**rdfp12a**, **rule9**) and transitive (**rdfp12c**, **rule10**), respectively. Therefore, the same machinery is adopted for them. The equivalency information is used in the extended structural subsumption algorithm as explained in Subsection 3.3.5. In this computation, the relation of subsumption of class/property individual is expanded to the equivalent group of the class/property.

DifferentFrom Pairs and DisjointWith Pairs

On the other hand, owl:differentFrom is reflexive but not transitive. Therefore, the pairwise relation is not resolved into one group. Each of a pair is registered to the other individual. The symbol $\asymp$ in the subsumption algorithm uses this information. Such machinery is the same for the class relation of owl:disjointWith.

If a class is disjoint with another class, the subclasses of the class are also disjoint with the subclasses of the other disjoint class. See **rule4** in Table 3.3, which is implemented in the function `owl-disjoint-p`. If disjoint classes are specified as multiple classes in an instance definition, the system signals an alarm of unsatisfiability. See **unsatisfiability3** and **4** in Table 3.1.

Functional Property

The entailment rule for an instance of owl:FunctionalProperty is described by **rdfp1** in [72]. The system maintains the bookkeeping of the inverse relation of a functional property in addition to the functional property itself. Then, the function `owl-same-p`, which is denoted as $\doteq$ in the explanation of the extended structural subsumption algorithm in Subsection 3.3.5, infers this equality of individuals.

The following shows an example of this entailments. See also $\simeq$ in the subsumption algorithm.

```
gx-user(2): (defProperty hasband
                (rdf:type owl:FunctionalProperty)
              (rdfs:domain Woman)
              (rdfs:range Man))
Warning: Range entailX3 by rdfs:domain: Woman rdf:type rdfs:Class.
Warning: Range entailX3 by rdfs:range: Man rdf:type rdfs:Class.
#<owl:FunctionalProperty hasband>
```

```
gx-user(3): (defIndividual MarieTherese (hasband LouisXIVdeFrance))
Warning: Range entailX3 by hasband: LouisXIVdeFrance rdf:type Man.
#<Woman MarieTherese>
gx-user(4): (defIndividual MarieTherese (hasband Roi-Soleil))
Warning: Range entailX3 by hasband: Roi-Soleil rdf:type Man.
#<Woman MarieTherese>
gx-user(5): (-> MarieTherese hasband)
(#<Man LouisXIVdeFrance> #<Man Roi-Soleil>)
gx-user(6): (owl-same-p LouisXIVdeFrance Roi-Soleil)
t
```

In this demonstration, `LouisXIVdeFrance` and `Roi-Soleil` are explicitly not stated as the same, but it is entailed by a property `hasband`, which is an instance of owl:FunctionalProperty.

Inverse Functional Property

For owl:InverseFunctionalProperty, the same machinery for entailment computation is made. See **rdfp2**. See also $\simeq$ in subsumption algorithm.

The following shows an entailment example of owl:InverseFunctionalProperty.

```
gx-user(7): (defProperty hasWife (rdf:type owl:InverseFunctionalProperty)
              (rdfs:domain Man)
              (rdfs:range Woman))
#<owl:InverseFunctionalProperty hasWife>
gx-user(8): (defIndividual Obama (hasWife Michelle))
Warning: Range entailX3 by hasWife: Michelle rdf:type Woman.
#<Man Obama>
gx-user(9): (defIndividual The44thPresidentOfUnitedStates
              (hasWife Michelle))
#<Man The44thPresidentOfUnitedStates>
gx-user(10): (owl-same-p Obama The44thPresidentOfUnitedStates)
t
```

Where `Obama` and `The44thPresidentOfUnitedStates` is not same explicitly, but SWCLOS entailed it because an inverse functional property `hasWife` supports it.

Symmetric Property

The symmetric property implicitly defines the same relation between an object and a subject as well as between the subject and the object. The system registers the symmetric relation to both ends upon one assertion. See also **rule8**.

The following shows an example of owl:SymmetricProperty.

```
gx-user(11): (defProperty spouse
                 (rdf:type owl:SymmetricProperty))
#<owl:SymmetricProperty spouse>
gx-user(12): (defIndividual Bill (spouse Hillary))
#<|rdfs:Resource| Bill>
gx-user(13): (-> Bill spouse)
#<|rdfs:Resource| Hillary>
gx-user(14): (-> Hillary spouse)
#<|rdfs:Resource| Bill>
```

Intersection of Concepts

The intersection of concepts turns out a subclass of each component. Namely, if $A \equiv C_1 \sqcap \cdots \sqcap C_n$ (where $i = 1, \ldots, n$), then $A \sqsubseteq C_i$. The system adds every class C_i to the *direct-superclasses list* of class A in CLOS from owl:intersectionOf assertions. Therefore, the subsumption computation is primarily performed by CLOS. In addition, the strong subsumption is computed by `subsumed-p` procedure. Note that if there is another class B that partly shares the component of the intersection, the CLOS class-superclass relation between A and the class B is adjusted ($A \preceq B$ or $B \preceq A$ according to the inclusiveness of the intersection sets).

Union of Concepts

Inversely, the union of concepts turns out a superclass of each component. Namely, if $A \equiv C_1 \sqcup \cdots \sqcup C_n$ (where $i = 1, \ldots, n$), then $C_i \sqsubseteq A$. The system adds class A to the direct-superclasses list of every class C_i in CLOS from owl:unionOf assertions. Therefore, the subsumption computation of owl:unionOf is performed by CLOS as well as of owl:intersectionOf. In addition, if there is another class B that partly shares the component of the union, the CLOS class-superclass relation between A and the class B is adjusted ($A \preceq B$ or $B \preceq A$ according to the inclusiveness of the union sets).

Complement of Concept

The complement relation is reflexive (see **rule5**) and entails disjointedness (**rule6**). The system registers each of the complement pair with the other for complementness and disjointness.

3.4 OWL Demonstration in SWCLOS

Cardinality checking

The following shows an example of entailments by the restriction of owl:allValuesFrom and owl:cardinality.

```
(defResource Wine
  (rdfs:subClassOf
    owl:Thing
    (owl:Restriction
      (owl:onProperty hasMaker)
      (owl:allValuesFrom Winery))
    (owl:Restriction
      (owl:onProperty hasColor)
      (owl:cardinality 1))))
-> #<rdfs:Class Wine>

(defResource Color (rdf:type owl:Class)
  (rdfs:subClassOf owl:Thing))
-> #<owl:Class Color>
(defIndividual Red (rdf:type Color))
-> #<Color Red>
(defIndividual White (rdf:type Color))
-> #<Color White>
(defIndividual MyHomeMadeWine
  (rdf:type Wine)
  (hasMaker MyHome))
-> #<Wine MyHomeMadeWine>
MyHome
-> #<Winery MyHome>
(defIndividual MyBlendedWine
  (rdf:type Wine)
  (hasColor Red White))
Error: cardinality constraint condition
  unsatisfiable: hasColor's max
  cardinality 1 unsatisfiable with
  (#<Color Red> #<Color White>).
```

Where `MyHome` is entailed as instance of `Winery`, because a value for `hasMaker` must be so by owl:allValuesFrom restriction, as defined in `Wine` definition. On the other hand, the definition of `MyBlendedWine` caused an error, because the cardinality restriction is inherited from `hasColor`.

3.5 Concluding Remarks

In this chapter, the semantic gap between object-centric CLOS and property-centric RDF(S) was filled by setting property resource objects in CLOS and inventing the collection mechanism for the property extension in RDF(S) through CLOS native slot-definition facilities. Although the CLOS type system is very close to RDFS semantics, the problem of membership loop at rdfs:Class arising from straightforward mapping of class/instance relationship was solved by inventing a proxy of rdfs:Class that has a twisted relation to rdfs:Class. The flexible implicit slot definition on demand was embodied in the class-based CLOS system. In order to accept forward-reference for entities, the novel functionality called *proactive entailment* was realized using RDF/OWL entailments. The domain and range constraint were developed with the property inheritance mechanism and embodied into SWCLOS.

It should be noted for readers unfamiliar with the practice of Semantic Web tools that the characteristics of SWCLOS is very different from other tools. SWCLOS was built so as to match the characteristics of interactive lisps. Programmers or ontologists can build ontologies interactively on top of lisp's Read-Eval-Print Loop (REPL) mode. Thus, if users input a piece of ontology into SWCLOS, SWCLOS immediately evaluates it, and performs default reasoning and satisfiability checking, whereas most of other tools read ontology files in batch mode with no warning for unsatisfiability, and then validation checking is invoked by users. It often results that most of people will suffer bunch of errors.

In order to realize the OWL universe in the RDF universe, only one axiom that owl:Class have to be a subclass of owl:Thing was added into the set of axioms, which are not stated in the OWL description file. All semantics and functionality of OWL specification was implemented on top of RDF(S) subsystem along with housekeeping facilities for OWL.

The efficiency of implementation is described the next chapter, and OWL Full programming is demonstrated after the next chapter.

Table 3.3: Entailment Rules added in OWL for SWCLOS

	If	Then
rule1a	*v p w*	
	v type Class	*v* subtype Thing
rule1b	*v p w*	
	w type Class	*w* subtype Thing
rule2a	*u* intersectionOf $\{v_j \ldots\}$	v_j type Class
rule2b	*u* unionOf $\{v_j \ldots\}$	v_j type Class
rule3	*x* distinctMembers $\{x_j \ldots\}$	x_j type Thing
rule4	*u* disjointWith *v*	
	u′ subClassOf *u*	*u′* disjointWith *v′*
	v′ subClassOf *v*	
rule5	*u* complementOf *v*	*v* complementOf *u*
rule6	*u* complementOf *v*	*v* disjointWith *u*
rule7	*u* oneOf $\{x_j \ldots\}$	x_j type *u*
rule8	*v* allValuesFrom *w*	
	v onProperty *p*	*w* subtype *u*
	p range *u*	
rule9	*p* type SymmetricProperty	
	p domain *C*	*C* equivalentClass *D*
	p range *D*	
rule10	*p* equivalentProperty *q*	*q* equivalentProperty *p*
rule11	*p* equivalentProperty *q*	
	q equivalentProperty *r*	*p* equivalentProperty *r*
rule12a	*p* equivalentProperty *q*	
	p domain *u*	*q* domain *u*
rule12b	*p* equivalentProperty *q*	
	p range *u*	*q* range *u*
rule13a	*p* inverseOf *q*	
	p domain *u*	*q* range *u*
rule13b	*p* inverseOf *q*	
	p range *u*	*q* domain *u*
rule14	*u* disjointWith *v*	
	a type *u*	*a* differentFrom *b*
	b type *v*	

[This page intentionally left blank]

Chapter 4

Benchmark Test by LUBM

The efficiency of implementation is tested by Lehigh University Benchmark (LUBM)[1], and SWCLOS showed the comparable performance to other OWL reasoners reported in Guo et al. [22]. SWCLOS replied with correct answers to all LUBM queries, whereas no other reasoners but OWLJessKB replied correctly to all queries. It should be noted that SWCLOS is not an application system for OWL ontology repository. It is an object oriented language for OWL modeling. All of ontology data are maintained on memory in SWCLOS. The results also showed the requirement of persistentization for instance objects for the purpose of the usage of ontology repository.

The rationale and elemental problems of benchmark tests for Semantic Web tools are also discussed in the discussion of related work in this chapter.

4.1 Characteristics of Lehigh University Benchmark

Generally, systems that are applicable to complex problems tends to be massive and complex. Therefore, it is difficult to manage ontologies against both complexity and scalability, but coping with them has been recently becoming a critical problem in Semantic Webs.

LUBM is designed to test the scalability of ABox or a set of instances. The ontology of university domain in LUBM is categorized to OWL Lite level, whereas the ABox, which can be artificially generated by a program, may be large and the sizes can be varied. Therefore, it is suitable to test ABox scalability for variable sizes of ABox as database where the underlying class schema (TBox) is shared by all sets of ABox test data. Therefore, it is not available to test the scalability of TBox.

[1] http://swat.cse.lehigh.edu/projects/lubm/

Thus, LUBM is not suitable to tableaux based systems like Racer[2] and Pellet[3], and other Description Logic based systems like KAON2[4], whereas these tools were tested and reported for LUBM by tool's developer themselves. As well, it should be noted that the aim of LUBM is not suitable to test SWCLOS, an object-oriented programming language for Semantic Webs. However, it is worth comparing SWCLOS to memory-based OWL reasoners reported by Guo, et al. [22] for two reasons. First, there is no other appropriate benchmark test reports yet, and second, it suggests the way of improvement of SWCLOS language system.

4.1.1 Characteristics of University Domain in LUBM

As mentioned above, LUBM domain ontology falls into OWL Lite. However, there is no entry for owl:sameAs and no owl:hasValue restriction. There are some OWL specific features as follows.

owl:TransitiveProperty: `ub:subOrganizationOf` is a transitive property.

owl:inverseOf: There are two inverse property relationships, <`ub:hasAlumnus`, `ub:degreeFrom`> and <`ub:memberOf`, `ub:member`>.

In addition, there are five subproperty relationships as follows.

1. `ub:doctoralDegreeFrom` is a subproperty of `ub:degreeFrom`.
2. `ub:mastersDegreeFrom` is a subproperty of `ub:degreeFrom`.
3. `ub:undergraduateDegreeFrom` is a subproperty of `ub:degreeFrom`.
4. `ub:headOf` is a subproperty of `ub:worksFor`.
5. `ub:worksFor` is a subproperty of `ub:memberOf`.

Therefore, the treatment of these properties is critical to obtain right answers for queries.

4.2 Queries for Benchmark Test in LUBM

Guo, et al. set up 14 queries for a university domain ontology. Most of them are for testing the extensionality, that is, queries for instances that satisfy some conditions. Each of queries are described below in simple SPARQL[5] forms and SWCLOS query programs, which are encoded so as

[2] `http://www.racer-systems.com/products/racerpro/index.phtml`
[3] `http://www.mindswap.org/2003/pellet/index.shtml`
[4] `http://kaon2.semanticweb.org/`
[5] `http://www.w3.org/TR/rdf-sparql-query/`

Thus, LUBM is not suitable to tableaux based systems like Racer[2] and Pellet[3], and other Description Logic based systems like KAON2[4], whereas these tools were tested and reported for LUBM by tool's developer themselves. As well, it should be noted that the aim of LUBM is not suitable to test SWCLOS, an object-oriented programming language for Semantic Webs. However, it is worth comparing SWCLOS to memory-based OWL reasoners reported by Guo, et al. [22] for two reasons. First, there is no other appropriate benchmark test reports yet, and second, it suggests the way of improvement of SWCLOS language system.

4.1.1 Characteristics of University Domain in LUBM

As mentioned above, LUBM domain ontology falls into OWL Lite. However, there is no entry for owl:sameAs and no owl:hasValue restriction. There are some OWL specific features as follows.

owl:TransitiveProperty: `ub:subOrganizationOf` is a transitive property.

owl:inverseOf: There are two inverse property relationships, <`ub:hasAlumnus`, `ub:degreeFrom`> and <`ub:memberOf`, `ub:member`>.

In addition, there are five subproperty relationships as follows.

1. `ub:doctoralDegreeFrom` is a subproperty of `ub:degreeFrom`.
2. `ub:mastersDegreeFrom` is a subproperty of `ub:degreeFrom`.
3. `ub:undergraduateDegreeFrom` is a subproperty of `ub:degreeFrom`.
4. `ub:headOf` is a subproperty of `ub:worksFor`.
5. `ub:worksFor` is a subproperty of `ub:memberOf`.

Therefore, the treatment of these properties is critical to obtain right answers for queries.

4.2 Queries for Benchmark Test in LUBM

Guo, et al. set up 14 queries for a university domain ontology. Most of them are for testing the extensionality, that is, queries for instances that satisfy some conditions. Each of queries are described below in simple SPARQL[5] forms and SWCLOS query programs, which are encoded so as

[2] `http://www.racer-systems.com/products/racerpro/index.phtml`
[3] `http://www.mindswap.org/2003/pellet/index.shtml`
[4] `http://kaon2.semanticweb.org/`
[5] `http://www.w3.org/TR/rdf-sparql-query/`

to simulate the behavior of SPARQL query language as much as possible.

Query 1. This query simply retrieves graduate students who take `GraduateCourse0` in `Department0` of `University0`. The computational time is less than $O(gurduatestudents)$. The number of graduate students are large but the selection condition is highly selective. In SPARQL and SWCLOS, this query is written as follows.

```
SELECT ?X
WHERE
  {?X rdf:type ub:GraduateStudent.
   ?X ub:takesCourse http://www.Department0.University0.edu/GraduateCourse0}
```

```
(defun q1 ()
  (loop for student in (collect-all-instances-of ub::GraduateStudent) with results
      when (member ub::Department0.University0.GraduateCourse0
                   (mklist (get-value student ub::takesCourse)))
      do (push student results)
      finally (return results)))
```

Query 2. This query requires a specific triangle relationship among three kinds of entities. In this case, the computational time in simple encoding will be $O(graduatestudents \times universities \times departments)$. It amounts up to $O(1874 \times 979 \times 15) = O(0.275 \times 10^8)$ even in LUBM(1,0). Therefore, the lisp code is modified simply to reduce the computational time.

```
SELECT ?X, ?Y, ?Z
WHERE
  {?X rdf:type ub:GraduateStudent.
   ?Y rdf:type ub:University.
   ?Z rdf:type ub:Department.
   ?X ub:memberOf ?Z.
   ?Z ub:subOrganizationOf ?Y.
   ?X ub:undergraduateDegreeFrom ?Y}
```

```
(defun q2.1 ()
  (let ((students (collect-all-instances-of ub::GraduateStudent))
        (universities (collect-all-instances-of ub::University)))
    (loop for student in students with results
        do
          (loop for department in (mklist (get-value student ub::memberOf))
              when (typep department ub::Department)
              do
                (loop for university in universities
                    when (and (member university
                                      (mklist (get-value student
```

```
                                                    ub::undergraduateDegreeFrom)))
                            (member university
                                    (mklist (get-value department
                                                    ub::subOrganizationOf))))
                  do (push (list student university department) results)))
        finally (return results))))
```

Query 3. This is analogous to Query 1 but the input data are rather small and the selection condition is broad.

```
SELECT ?X
WHERE
  {?X rdf:type ub:Publication.
   ?X ub:publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0}
```

```
(defun q3 ()
  (loop for publication in (collect-all-instances-of ub::Publication) with results
      when (member ub::Department0.University0.AssistantProfessor0
                   (mklist (get-value publication ub::publicationAuthor)))
      do (push publication results)
      finally (return results)))
```

Query 4. This question looks like requiring large amount of computational time of $O(professors \times names \times emailaddresses \times telephones)$. However, supposing the number of names, emailaddresses, and telephones are one, it becomes $O(professors)$. Note that this question assumes rdfs:subClassOf relationship between `ub:Professor` and its subclasses.

```
SELECT ?X, ?Y1, ?Y2, ?Y3
WHERE
  {?X rdf:type ub:Professor.
   ?X ub:worksFor <http://www.Department0.University0.edu>.
   ?X ub:name ?Y1.
   ?X ub:emailAddress ?Y2.
   ?X ub:telephone ?Y3}
```

```
(defun q4 ()
  (loop for professor in (collect-all-instances-of ub::Professor) with results
      when (member ub::Department0.University0.
                   (mklist (get-value professor ub::worksFor)))
      do (loop for name in (mklist (get-value professor ub::name))
               do
                 (loop for email in (mklist (get-value professor ub::emailAddress))
                     do
                       (loop for tel in (mklist (get-value professor ub::telephone))
                           do (push (list professor name email tel)
```

```
                                  results))))
      finally (return results)))
```

Query 5. This question looks simple. However, it assumes the combination of class hierarchy of ub:Person and property hierarchy of ub:memberOf. So, it is internally complex in inference.

```
SELECT ?X
WHERE
  {?X rdf:type ub:Person.
   ?X ub:memberOf <http://www.Department0.University0.edu>}

(defun q5 ()
  (loop for person in (collect-all-instances-of ub::Person) with results
      when (member ub::Department0.University0.
                   (mklist (get-value person ub:memberOf)))
      do (push person results)
      finally (return results)))
```

Query 6. This question looks quite simple described below.

```
SELECT ?X
WHERE
  {?X rdf:type ub:Student}

(defun q6 ()
  (collect-all-instances-of ub::Student))
```

However, it assumes the implicit super/subclass relation between ub:GraduateStudent and ub:Student in addition to the explicit one between ub:UndergraduateStudent and ub:Student. ub:Student is defined as an intersection of ub:Person and the restriction of ub:Course on ub:takesCourse. By contrast, ub:GraduateStudent is defined as a subclass of both ub:Person and restriction of ub:GraduateCourse on ub:takesCourse. See the following definitions.

```
(owl:Class ub:Student (rdfs:label "student")
 (owl:intersectionOf ub:Person
                     (owl:Restriction (owl:onProperty ub:takesCourse)
                                      (owl:someValuesFrom ub:Course))))

(owl:Class ub:GraduateStudent (rdfs:label "graduate student")
 (rdfs:subClassOf ub:Person
                  (owl:Restriction (owl:onProperty ub:takesCourse)
                                   (owl:someValuesFrom ub:GraduateCourse))))
```

Although there are no explicit relation between ub:Student and ub:GraduateStudent in the semantics of RDF, in OWL ub:Student is a superclass of ub:GraduateStudent, because a person who takes a ub:Course is a student due to owl:intersectionOf complete relation, and ub:GraduateStudent is a person who takes ub:GraduateCourse that is a subclass of ub:Course. See the following demonstration example in SWCLOS. The systems must infer these implicit subsumption combined with another super/subclass relation.

```
gx-user(4): (rdf-subtypep ub:GraduateStudent ub:Student)
nil
t
gx-user(5): (subsumed-p ub:GraduateStudent ub:Student)
t
t
```

Query 7. This query is for students who take courses by AssociateProfessor0 of Department0 in University0. This query includes a query for matching a constant subject in triple.

```
SELECT ?X, ?Y
WHERE
  {?X rdf:type ub:Student.
   ?Y rdf:type ub:Course.
   ?X ub:takesCourse ?Y.
   <http://www.Department0.University0.edu/AssociateProfessor0> ub:teacherOf ?Y}
```

```
(defun q7 ()
  (loop for student in (collect-all-instances-of ub::Student) with results
      and courses = (mklist (get-value
                             ub::Department0.University0.AssociateProfessor0
                             ub::teacherOf))
      do
        (loop for course in courses
            when (member course (mklist (get-value student ub::takesCourse)))
            do (push (list student course) results))
      finally (return results)))
```

Query 8. This question is similar Query 7 rather than Query 2, and more complex than Query 7 by adding one more property. However, SWCLOS results very differently than other reasoners as shown below. The analysis of this distinction will be presented at a subsection below.

```
SELECT ?X, ?Y, ?Z
WHERE
  {?X rdf:type ub:Student.
   ?Y rdf:type ub:Department.
```

```
   ?X ub:memberOf ?Y.
   ?Y ub:subOrganizationOF <http://www.University0.edu>.
   ?X ub:emailAddress ?Z}

(defun q8 ()
  (let ((students (collect-all-instances-of ub::Student)
        (departments (collect-all-instances-of ub::Department))))
    (loop for department in departments with results
        when (member ub::University0.
                     (mklist (get-value department ub::subOrganizationOf)))
        do
          (loop for student in students
              when (member department
                           (mklist (get-value student ub::memberOf)))
              do
                (loop for email in (mklist (get-value student ub::emailAddress))
                    do (push (list student department email)
                             results)))
        finally (return results))))
```

Query 9. This query handles a triangle relationship among three kinds of entities like Query 2. In this case, the straightforward encoding from the SPARQL query form produces $O(students \times faculties \times courses)$. Even in LUBM(1,0) it is up to $O(7790 \times 1627 \times 540) = O(0.68 \times 10^{10})$. However, refactoring the straightforward encoding for student's advisor is easy to obtain the efficiency. Therefore, the lisp code is modified apart from straightforward mapping from SPARQL.

```
SELECT ?X, ?Y, ?Z
WHERE
  {?X rdf:type ub:Student.
   ?Y rdf:type ub:Faculty.
   ?Z rdf:type ub:Course.
   ?X ub:advisor ?Y.
   ?Y ub:teacherOf ?Z.
   ?X ub:takesCourse ?Z}

(defun q9.1 ()
  (loop for student in (collect-all-instances-of ub::Student) with results
      do
        (loop for faculty in (mklist (get-value student ub:advisor))
            when (typep faculty ub:Faculty)
            do
              (loop for course in (mklist (get-value faculty ub:teacherOf))
                  when (and (typep course ub:Course)
                            (member course
                                    (mklist
                                      (get-value student ub:takesCourse))))
```

```
                do (push (list student faculty course)
                         results)))
      finally (return results)))
```

Query 10. This query is simpler than Query 7, but it requires the implicit super/subclass relationship between ub:GraduateStudent and ub:Student as well as Query 6. Additionally, it is more complex than Query 6 by query for one more property.

```
SELECT ?X
WHERE
  {?X rdf:type ub:Student.
   ?X ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>}

(defun q10 ()
  (loop for student in (collect-all-instances-of ub::Student)
      when (member ub::Department0.University0.GraduateCourse0
                   (mklist (get-value student ub::takesCourse)))
      collect student))
```

Query 11. This query form is similar to Query 1. The number of entities are small, but this query assumes the transitive property of ub:subOrganizationOf. Note that subsumed-p in SWCLOS computes the transitivity of individuals on transitive properties. See Subsection 3.3.5.

```
SELECT ?X
WHERE
  {?X rdf:type ub:ResearchGroup.
   ?X ub:subOrganizationOf <http://www.University0.edu>}

(defun q11 ()
  (loop for research-group in (collect-all-instances-of ub::ResearchGroup)
      when (some #'(lambda (x) (subsumed-p x ub::University0.))
                 (mklist (get-value research-group ub::subOrganizationOf)))
      collect research-group))
```

Query 12. This query is highly OWL specific. ub:Chair has no direct instances and no direct subclasses. Therefore, the class extension is implicit. Note that ub:Chair is defined as a subclass of ub:Professor and it is defined such as a chair is a professor who has a property ub:headOf for some ub:Department. Thus, the system must infer that a professor who has a head property of some department is an instance of class ub:Chair.

```
SELECT ?X, ?Y
WHERE
```

```
  {?X rdf:type ub:Chair.
   ?Y rdf:type ub:Department.
   ?X ub:worksFor ?Y.
   ?Y ub:subOrganizationOf <http://www.Department0.edu>}

(defun q12 ()
  (loop for chair in (collect-all-instances-of ub::Chair) with results
      do
        (loop for department in (mklist (get-value chair ub::worksFor))
            when (and (typep department ub::Department)
                      (member ub::University0.
                              (mklist
                                (get-value department ub::subOrganizationOf))))
            do (push (list chair department) results))
      finally (return results)))
```

Query 13. This query assumes the combination of owl:inverseOf and rdfs:subPropertyOf constructs. Note that there is no explicit property extensions of ub:hasAlumnus. However, ub:hasAlumnus is an inverse of ub:degreeFrom. Furthermore, ub:degreeFrom is a superproperty of three properties, ub:doctoralDegreeFrom, ub:mastersDegreeFrom, and ub:undergraduateDegreeFrom. The system must infer these relationship correctly.

```
SELECT ?X
WHERE
  {?X rdf:type ub:Person.
   <http://www.University0.edu> ub:hasAlumnus ?X}

(defun q13 ()
  (loop for person in (collect-all-instances-of ub:Person) with results
      and values = (mklist (get-value ub::University0. ub::hasAlumnus))
      do
        (loop for val in values
            when (rdf-equalp person val)
            do (push person results))
      finally (return results)))
```

Query 14. This query is the simplest in the test query set. It assumes no OWL specific features and no RDFS hierarchical structures. Note that ub:UndergraduateStudent has no subclasses. Thus, in this case, SWCLOS only retrieve the pointer to the list of direct instances of ub:UndergraduateStudent.

```
SELECT ?X
WHERE
```

Table 4.1: LUBM Benchmark Loading Time (dd:hh:mm:ss)

-	Triples #	OWLJessKB-NP	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
LUBM(1,0)	103,397	00:02:19:18	00:00:11:31	00:00:06:36	00:00:03:06	00:00:01:43
LUBM(5,0)	646,128	-	00:27:14:00	00:15:26:00	00:08:42:00	00:04:54:00
LUBM(10,0)	1,316,993	-	08:13:00:00	05:03:00:00	02:18:00:00	01:13:00:00

OWLJessKB-NP: Pentium 4 CPU 1.8GHz, 256MB RAM, Windows XP Professional
SWCLOS CPU1: Pentium 4 CPU 1.50GHz, 256MB RAM, Windows XP Home Edition SP2
SWCLOS CPU2: Celeron CPU 2.53GHz, 512MB RAM, Windows XP Professional SP3
SWCLOS CPU3: Core 2 1.66+1.66GHz, 0.99GB RAM, Windows XP Professional SP3
SWCLOS CPU4: Core 2 2.93+2.93GHz, 1.94GB RAM, Windows XP Professional SP3

```
  {?X rdf:type ub:UndergraduateStudent}

(defun q14 ()
  (collect-all-instances-of ub::UndergraduateStudent))
```

4.3 Experimental Results

4.3.1 Loading of LUBM

The loading times of LUBM are summarized in Table 4.1 with the size of LUBM ontologies and the loading time of OWLJessKB-NP, which is a non-persistent version of OWLJessKB. Note that OWLJessKB is only one that replied correct answers for all queries in the LUBM report [22] and also note that the CPU clock and memory size for OWLJessKB-NP is different from SWCLOS CPU1, CPU2, CPU3, and CPU4. However, it is reasonable to compare the performance of OWLJessKB to an intermediate value between CPU1 and CPU2 by reason of their clock and memory size.

OWLJessKB-NP takes approximately 10 times to load LUBM(1,0) ontology than SWCLOS. Furthermore, OWLJessKB cannot load LUBM(5,0) and LUBM(10,0). The other reasoners than OWLJessKB in the LUBM report can load larger ontologies, LUBM(10,0) and LUBM(20,0), but all of them cannot answer correctly for some queries. See the detail in Guo et al. [22]. In case of SWCLOS, LUBM(5,0), and LUBM(10,0) are also loadable even for CPU1.

4.3.2 Results for Queries

The results for queries are listed at Table 4.2 for LUBM(1,0), Table 4.3 for LUBM(5,0), and Table 4.4 for LUBM(10,0) with the report data in [22] for the comparison. Sesami fails to answer for Query 6 to 13. It suggests that Sesami cannot be equipped for highly OWL specific functionalities such as implicit extensionality. DLDB-OWL fails to answer for Query 11 to 13. It might implies that inference on transitive property and super/sub property are not correctly devised.

Note that generally DLDB-OWL (databased) and Sesami (memory-based and databased) are faster than OWLJessKB, but those systems do not reply correctly for queries. It deserves to note that the computational capability generally shows tradeoff relationship to computational efficiency. Therefore, OWLJessKB-NP (memory-based) is suitable to compare the efficiency with SWCLOS, because both are fully equipped for inference that LUBM requires. The CPU speed of the machine used in the tests is different among SWCLOS and [22], but it is reasonable that Guo's numbers is compared to an intermediate value between SWCLOS CPU1 and CPU2 by their CPU clock and memory size.

Roughly speaking, the test results showed the computational efficiency of SWCLOS is comparable to other tools, especially in comparison to OWLJessKB-NP. For LUBM(5.0) and LUBM(10,0), OWLJessKB was not tested, since it failed to load test data. Note that at the time of [22] reported Racer cannot load LUBM data.[6] Note that we had no expansion of heap memory of Allegro Common Lisp 8.1.

Meanwhile, the results of SWCLOS showed some distinctive results about Query 2, 5, 8, and 13 in comparison with other queries. We analyze the reason of the distinctive behavior of SWCLOS in the next subsection.

4.3.3 Analysis of Distinctive Results

Refactoring query codes

Generally, more query variables in SPARQL forms, more complex query codes. However, in case no dependency among query variables, it amounts to actually simple codes, even if the code includes many loops like Query 4. Exactly, the complexity of query form in Query 4 is the same as Query 1. In both, substantial task is finding out ?x's bound values that satisfy the given conditions. By contrast, higher dependent relations such as Query 2 and Query 9, which show the triangle relations among query variables, are exactly more complex.

[6] It was tested for Racer by developer themselves. See the section 4.5 and Haarslev et al. [23]

Table 4.2: LUBM(1,0) Benchmark Test Results

Query	unit	DLDB -OWL	Sesame -DB	Sesame -Memory	OWL JessKB-P	OWL JessKB-NP	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
1	ms	59	46	15	9,203	200	83	63	27	16
	ans	4	4	4	4	4	4	4	4	4
2	ms	181	51,878	87	11,6297	3,978	75,154	54,818	21,328	12,047
	ans	0	0	0	0	0	0	0	0	0
3	ms	218	40	0	13,990	164	250	187	73	31
	ans	6	6	6	6	6	6	6	6	6
4	ms	506	768	6	211,514	8,929	547	412	177	109
	ans	34	34	34	34	34	34	34	34	34
5	ms	617	2,945	17	5,929	475	22,146	14,281	9,063	5,125
	ans	719	719	719	719	719	719	719	719	719
6	ms	481	253	48	1,271	112	52	47	21	16
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790	7,790	7,790
7	ms	478	603	3	128,115	67	1,583	1,141	432	250
	ans	67	59	59	67	67	67	67	67	67
8	ms	765	105,026	273	164,106	4,953	285,291	182,922	120,370	68,109
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790	7,790	7,790
9	ms	634	3,4034	89	87,475	2,525	974	703	281	156
	ans	208	103	103	208	208	208	208	208	208
10	ms	98	20	1	141	4	443	323	130	78
	ans	4	0	0	4	4	4	4	4	4
11	ms	48	65	1	1,592	45	62	41	16	16
	ans	0	0	0	224	224	224	224	224	224
12	ms	62	4,484	12	11,266	162	521	381	172	94
	ans	0	0	0	15	15	15	15	15	15
13	ms	200	4	1	90	1	15,427	11,875	5,021	2,812
	ans	0	0	0	1	1	1	1	1	1
14	ms	187	218	42	811	20	5	0	0	0
	ans	5,916	5,916	5,916	5,916	5,916	5,916	5,916	5,916	5,916

ms: milli-second, ans: number of answers
SWCLOS CPU1: Pentium 4 CPU 1.50GHz, 256MB RAM, Windows XP Home Edition SP2
SWCLOS CPU2: Celeron CPU 2.53GHz, 512MB RAM, Windows XP Professional SP3
SWCLOS CPU3: Intel Core 2 1.66+1.66GHz, 0.99GB RAM, Windows XP Professional SP3
SWCLOS CPU4: Intel Core 2 2.93+2.93GHz, 1.94GB RAM, Windows XP Professional SP3
not SWCLOS: Pentium4 CPU 1.80GHz, 256MB RAM, Windows XP Professional

Table 4.3: LUBM(5,0) BenchMark Test Results

Query	unit	DLDB -OWL	Sesame -DB	Sesame -Memory	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
1	ms	226	43	37	765	438	188	109
	ans	4	4	4	4	4	4	4
2	ms	2,320	368,423	495	484,172	345,032	129,375	72,703
	ans	9	9	9	9	9	9	9
3	ms	2,545	53	1	1,515	1,093	375	219
	ans	6	6	6	6	6	6	6
4	ms	2,498	823	4	3,296	2,391	1,063	609
	ans	34	34	34	34	34	34	34
5	ms	4,642	3,039	17	238,735	154,562	95,641	53,704
	ans	719	719	719	719	719	719	719
6	ms	4,365	1,517	251	391	266	125	63
	ans	48,582	36,682	36,682	48,582	48,582	48,582	48,582
7	ms	2,639	636	4	9,234	6,797	2,375	1,344
	ans	67	59	59	67	67	67	67
8	ms	3,004	108,384	262	3,179,687	2,050,579	1,294,375	737,453
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790
9	ms	7,751	256,770	534	5,782	4,407	1,484	859
	ans	1,245	600	600	1,245	1,245	1,245	1,245
10	ms	1,051	36	0	2,594	1,921	735	406
	ans	4	0	0	4	4	4	4
11	ms	51	73	1	422	297	110	63
	ans	0	0	0	224	224	224	224
12	ms	78	4,659	14	3,156	2,375	1,016	594
	ans	0	0	0	15	15	15	15
13	ms	2,389	9	1	333,969	210,843	94,484	52,344
	ans	0	0	0	1	1	1	1
14	ms	2,937	1,398	257	31	31	0	16
	ans	36,682	36,682	36,682	36,682	36,682	36,682	36,682

ms: milli-second, ans: number of answers
SWCLOS CPU1: *ibid.*
SWCLOS CPU2: *ibid.*
SWCLOS CPU3: *ibid.*
SWCLOS CPU4: *ibid.*
not SWCLOS: *ibid.*

Table 4.4: LUBM(10,0) BenchMark Test Results

Query	unit	DLDB -OWL	Sesame -DB	Sesame -Memory	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
1	ms	412	40	106	1,172	922	344	203
	ans	4	4	4	4	4	4	4
2	ms	14,556	711,678	1,068	1,009,563	710,781	280,141	154,516
	ans	28	28	28	28	28	28	28
3	ms	5,540	59	0	3,079	2,156	734	407
	ans	6	6	6	6	6	6	6
4	ms	5,615	762	4	6,610	4,937	2,094	1,203
	ans	34	34	34	34	34	34	34
5	ms	11,511	3,214	17	758,391	472,328	302,953	167,250
	ans	719	719	719	719	719	719	719
6	ms	11,158	3,539	543	781	546	235	125
	ans	99,566	75,547	75,547	99,566	99,566	99,566	99,566
7	ms	7,028	5,916	4	18,656	13,437	4,516	2,531
	ans	67	59	59	67	67	67	67
8	ms	5,937	108851	264	10,084,219	6,391,422	4,229,938	2,322,890
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790
9	ms	19,971	460,267	1,123	11,344	8,421	2,953	1,657
	ans	2,540	1,233	1,233	2,540	2,540	2,540	2,540
10	ms	2,339	40	0	5,750	3,812	1,344	765
	ans	4	0	0	4	4	4	4
11	ms	61	84	3	875	562	281	125
	ans	0	0	0	224	224	224	224
12	ms	123	4,703	12	6,469	4,719	2,062	1,171
	ans	0	0	0	15	15	15	15
13	ms	5,173	12	1	1,953,141	764,141	363,000	197,406
	ans	0	0	0	1	1	1	1
14	ms	7,870	3,831	515	78	63	31	15
	ans	75,547	75,547	75,547	75,547	75,547	75,547	75,547

ms: milli-second, ans: number of answers
SWCLOS CPU1: *ibid.*
SWCLOS CPU2: *ibid.*
SWCLOS CPU3: *ibid.*
SWCLOS CPU4: *ibid.*
not SWCLOS: *ibid.*

In the experiment of benchmarking described above, the straightforward mapping from SPARQL forms may cause huge amount of computational time. Therefore, such codes are simply modified by looking the textual expressions. For example, Suppose we obtained an code such that;

```
(defun query ()
  (let ((xx (collect-all-instances-of ClassX))
        (yy (collect-all-instances-of ClassY))
        (zz (collect-all-instances-of ClassZ)))
    (loop for x in xx with results
      do (loop for y in yy
           do (loop for z in zz
                do (when (and --satisify-given-condition-for-x--
                              --satisify-given-condition-for-y--
                              --satisify-given-condition-for-z-- )
                     (push (list x y z) results))))
      finally (return results)))).
```

In case of finding `yy` only depends `xx` in `-satisify-given-condition-for-y-`, it may be tailored such as;

```
(defun query ()
  (let ((xx (collect-all-instances-of ClassX))
        (zz (collect-all-instances-of ClassZ)))
    (loop for x in xx with results
      do (loop for y in --satisify-given-condition-for-y--
           do (loop for z in zz
                do (when (and --satisify-given-condition-for-x--
                              --satisify-given-condition-for-z-- )
                     (push (list x y z) results))))
      finally (return results)))).
```

This tailoring reduces the computational time to $O(n_1 \times n_2' \times n_3)$ from $O(n_1 \times n_2 \times n_3)$, where n_1, n_2, and n_3 is a number of entities of three kinds, respectively, and n_2' is a number of satisfiable entities for the given condition. Thus, the lisp codes of Query 2 and 9 are simply modified in order to reduce the computational time. See two lisp codes `q2.1` and `q9.1` described above.

For Query 2, we might be able to furthermore improve the efficiency. Fig.4.1 depicts the triangle structure in Query 2. As shown here, an instance of `ub:GraduateStudent` will be a subject for not only `ub:memberOf` triple but also `ub:underGraduateFrom` triple, and there is a dependency of a student to a department + a department to a university. Therefore, we can make a match between two computational values for universities against a student and a department. The following code `q2.2` is a newly modified code for Query 2 for pursuing the dependency.

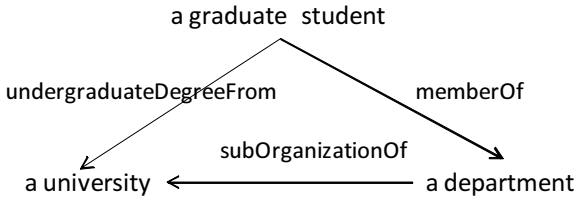

Fig. 4.1: Triangle Structure of Q2.

```
(defun q2.2 ()
  (loop for student in (collect-all-instances-of ub::GraduateStudent) with results
      do
        (loop for department in (gx::mklist (get-value student ub::memberOf))
            when (typep department ub::Department)
            do
              (loop for university in (gx::mklist (get-value department ub::subOrganizationOf))
                  do
                    (when (member university
                                  (gx::mklist (get-value student ub::undergraduateDegreeFrom)))
                      (push (list student university department) results))))
      finally (return results)))
```

As well as Query 2, Query 8 can be analyzed from the data dependency. The code `q8` listed above for Query 8 was distorted from straightforward encoding. The structure of Query 8 is depicted in Fig. 4.2. Then, we could improve the efficiency by pursuing regularly dependency from students to departments. The following shows the dependency code for Query 8.

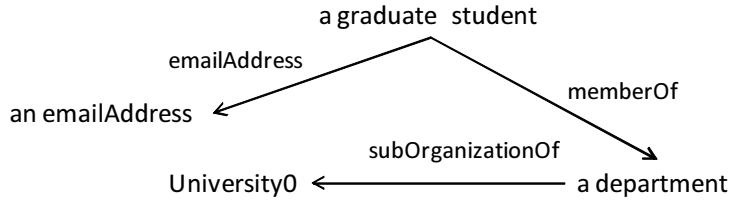

Fig. 4.2: Triangle Structure of Q8.

```
(defun q8.1 ()
  (let ((students (collect-all-instances-of ub::Student))
        (departments (collect-all-instances-of ub::Department)))
    (loop for student in students with results
        do
          (loop for department in (gx::mklist (get-value student ub::memberOf))
              when (typep department ub::Department)
              do
```

Table 4.5: Results of Refactoring Lisp Query Code for LUBM(1,0)

Query	unit	DLDB -OWL	Sesame -DB	Sesame -Memory	OWL JessKB-P	OWL JessKB-NP	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
2	ms	181	51,878	87	11,6297	3,978	4,844	3,328	2,062	1,321
	ans	0	0	0	0	0	0	0	0	0
8	ms	765	105,026	273	164,106	4,953	19,500	13,297	8,781	5,407
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790	7,790	7,790

ms: milli-second, ans: number of answers
SWCLOS CPU1: Pentium 4 CPU 1.50GHz, 256MB RAM, Windows XP Home Edition SP2
SWCLOS CPU2: Celeron CPU 2.53GHz, 512MB RAM, Windows XP Professional SP3
SWCLOS CPU3: Core 2 1.66+1.66GHz, 0.99GB RAM, Windows XP Professional SP3
SWCLOS CPU4: Core 2 2.93+2.93GHz, 1.94GB RAM, Windows XP Professional SP3
not SWCLOS: Pentium4 CPU 1.80GHz, 256MB RAM, Windows XP Professional

```
            (when (member ub::University0.
                          (gx::mklist (get-value department ub::subOrganizationOf)))
              (loop for email in (gx::mklist (get-value student ub::emailAddress))
                  do (push (list student department email) results))))
      finally (return results))))
```

The results of this refactoring for LUBM(1,0) are summarized in Table 4.5. The computational time is reduced more than 10 times in comparison with Table 4.2.

Backpointers of predicates and objects

As described above, the improvement for efficiency was obtained by manual refactoring for Query 2 and 8 by pursuing the data dependency in triples. However, it is difficult that machines analyze structures of SPARQL query forms and automatically compile them and produce efficient lisp codes. In case of Query 2 and 8, if we have backpointers on subject/predicate/object triples, machines can easily compile SPARQL query forms without dependency analysis, since we can obtain any subject values associated to any objects with respect to some predicate. Eventually, it makes easy to produce efficient codes. The following code of CLOS method automatically adds backpointers from an object to subjects in SWCLOS with respect to every predicate that points the object from the subjects. Then, `q2` and `q8` may be changed to `qq2` and `qq8`, respectively by leveraging these backpointers.

```
(defmethod shared-initialize :after ((instance gnode) slot-names &rest initargs)
  (declare (ignore slot-names initargs))
  (typecase instance
    (rdfs:Datatype nil)
    (rdfs:Literal nil)
```

```
    (owl:Restriction nil)
    (rdfs:Resource
     (let* ((class (class-of instance))
            (slotds (mop:class-slots class)))
       (dolist (slotd slotds)
         (let ((slot-name (mop:slot-definition-name slotd)))
           (when (slot-boundp instance slot-name)
             (let ((val (slot-value instance slot-name)))
               (typecase val
                 (rdfs:Datatype nil)
                 (rdfs:Literal nil)
                 (owl:Restriction nil)
                 (rdfs:Resource
                  (let* ((inv-plist (slot-value val 'inv-plist))
                         (inv-vals (getf inv-plist slot-name)))
                    (cond (inv-vals (pushnew instance inv-vals))
                          (t (setq inv-vals (list instance))))
                    (setf (getf inv-plist slot-name) inv-vals)
                    (setf (slot-value val 'inv-plist) inv-plist)))))))))))

(defun qq2 ()
  (loop for university in (collect-all-instances-of ub::University)
      as students =
        (remove-if-not #'(lambda (x) (typep x ub::GraduateStudent))
                       (getf (slot-value university 'gx::inv-plist)
                             'ub::undergraduateDegreeFrom))
      as departments =
        (remove-if-not #'(lambda (x) (typep x ub::Department))
                       (getf (slot-value university 'gx::inv-plist)
                             'ub::subOrganizationOf))
      append
        (loop for student in students
            as belong = (get-value student ub:memberOf)
            when (member belong departments)
            collect (list student university belong))))

(defun qq8 ()
  (let ((departments
          (remove-if-not #'(lambda (x) (typep x ub::Department))
            (getf (slot-value ub::University0. 'gx::inv-plist) 'ub::subOrganizationOf))))
    (loop for department in departments
        append
          (let ((students
                  (remove-if-not #'(lambda (x) (typep x ub::Student))
                    (getf (slot-value department 'gx::inv-plist) 'ub::memberOf))))
            (loop for student in students
                append
                  (loop for email in (gx::mklist (get-value student ub::emailAddress))
```

Table 4.6: Query Results by Backpointer for LUBM(1,0)

Query	unit	DLDB -OWL	Sesame -DB	Sesame -Memory	OWL JessKB-P	OWL JessKB-NP	SWCLOS CPU1	SWCLOS CPU2	SWCLOS CPU3	SWCLOS CPU4
2	ms	181	51,878	87	11,6297	3,978	4,688	3,172	1,906	1,125
	ans	0	0	0	0	0	0	0	0	0
8	ms	765	105,026	273	164,106	4,953	390	250	94	63
	ans	7,790	5,916	5,916	7,790	7,790	7,790	7,790	7,790	7,790

ms: milli-second, ans: number of answers
SWCLOS CPU1: Pentium 4 CPU 1.50GHz, 256MB RAM, Windows XP Home Edition SP2
SWCLOS CPU2: Celeron CPU 2.53GHz, 512MB RAM, Windows XP Professional SP3
SWCLOS CPU3: Core 2 1.66+1.66GHz, 0.99GB RAM, Windows XP Professional SP3
SWCLOS CPU4: Core 2 2.93+2.93GHz, 1.94GB RAM, Windows XP Professional SP3
not SWCLOS: Pentium4 CPU 1.80GHz, 256MB RAM, Windows XP Professional

```
collect (list student department email)))))))
```

These codes, `qq2` and `qq8`, might look messy but they are actually simple and may be rearranged by introducing appropriate internal functions. It is easier for machines to produce such lisp codes from SPARQL forms. The results of these queries by leveraging the backpointers are summarized in Table 4.6. As you can see, it is approximately 1000 times efficient for Query 8 than Table 4.2. This drastic efficiency improvement is obtained by firstly getting the subject value associated to constant object `ub:University0`. In such a case, generally $O(n_1 \times n_2 \prime \times n_3)$ is reduced to $O(n_1 \times 1 \times n_3)$.

Memoization for Data Caching

In case such that query forms are simple but the inferences are complex, refactoring is not effective. Fig. 4.3 shows the result of analysis of Query 5 by Allegro Common Lisp Code Analyzer. As shown here, 90% of computational time is spent for the execution of `collect-all-extensions-of`. We know that LUBM ontology does not change in the query process. Therefore, memoization[7](caching results of function execution) will be effective. The memoization process is very simple. It only needs to invoke function `memoize` for the function to be memoized before the function execution. The result of memoization of function `collect-all-extensions-of` is summarized in Table 4.7 for all of queries in LUBM(1,0). The query execution was carried out twice successively. The execution time of Query 5 is drastically reduced in comparison with Table 4.2. The time for Query 13 is drastically reduced from the 1st execution to 2nd execution.

[7]Not memorization. For example, see *"Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp"*(Peter Norvig).

Table 4.7: Results of Memoization of `collect-all-extensions-of` for LUBM(1,0)

Query	unit	OWL JessKB-NP	CPU1 (1st/2nd)	CPU2 (1st/2nd)	CPU3 (1st/2nd)	CPU4 (1st/2nd)
1	ms	200	78/78	78/63	15/15	0/16
	ans	4	4	4	4	4
2	ms	3,978	69,468/69,531	53,469/54,391	19,563/19,562	10,750/10,828
	ans	0	0	0	0	0
3	ms	164	250/250	188/172	63/78	47/47
	ans	6	6	6	6	6
4	ms	8,929	531/547	406/422	172/188	94/109
	ans	34	34	34	34	34
5	ms	475	2,546/2,531	2,031/2,000	843/829	469/469
	ans	719	719	719	719	719
6	ms	112	47/47	47/47	31/15	0/16
	ans	7,790	7,790	7,790	7,790	7,790
7	ms	67	1,547/1,547	1,204/1,219	422/453	250/250
	ans	67	67	67	67	67
8	ms	4,953	11,235/11,204	8,734/8,797	3,360/3,360	1,875/1,875
	ans	7,790	7,790	7,790	7,790	7,790
9	ms	2,525	953/953	750/782	281/297	156/157
	ans	208	208	208	208	208
10	ms	4	422/422	343/375	125/141	78/78
	ans	4	4	4	4	4
11	ms	45	62/62	31/47	16/16	15/15
	ans	224	224	224	224	224
12	ms	162	515/515	375/390	156/172	93/94
	ans	15	15	15	15	15
13	ms	1	15,312/1,860	11,157/1,453	5,032/625	2,859/359
	ans	1	1	1	1	1
14	ms	20	0/0	0/0	16/0	0/0
	ans	5,916	5,916	5,916	5,916	5,916

ms: mili-second, ans: number of answers
SWCLOS CPU1: *ibid.*
SWCLOS CPU2: *ibid.*
SWCLOS CPU3: *ibid.*
SWCLOS CPU4: *ibid.*
not SWCLOS: *ibid.*

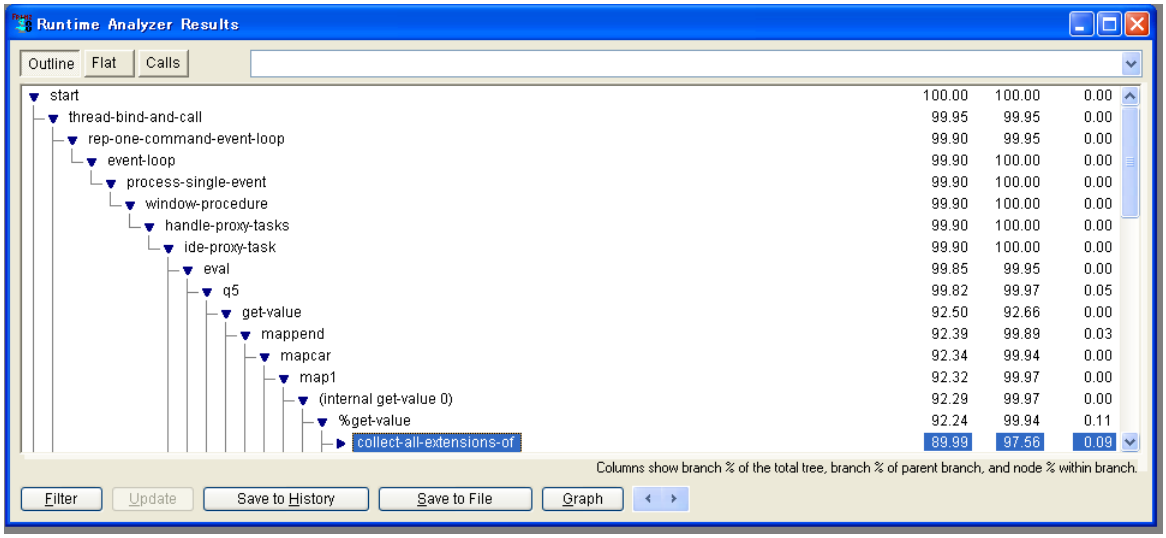

Fig. 4.3: Code Analysis of Query 5

Table 4.8: Summary of Analysis and Improvements

Improvement	Q2	Q5	Q8	Q13	note
refactoring	○	-	○	-	Effective SPARQL implementation is needed.
backpointer	○	-	○	-	Larger memory size is needed.
memoization	-	○	○	○	Applicable only for immutable ontologies

4.4 Summary of Analysis and Improvement for LUBM

The results of analysis and improvement of efficiency can be summarized at Table 4.8, which shows the effectiveness of improvements for Query 2, 5, 8 and 13, which initially demonstrated distinctive behaviors in computational efficiency.

Manual refactoring is effective for queries such as involve combinatorial explosions in SPARQL query forms. However, refactoring is not generally applicable for all queries and it is difficult to automatically produce efficient codes for SPARQL queries.

Backpointer is an alternative solution for combinatorial explosions but it requires larger memory size instead of obtaining drastically efficient results. For example, in LUBM(10,0), additional 10MB memory is required for backpointer memory cells, since one backpointer requires two cons cells of 4 memory words.

Memoization is widely effective from inferential complexity problems for OWL reasoning to combinatorial complexity problems in query forms, but it is not applicable to mutable ontologies, and it will require a special code to clear the memoized data. It is also difficult to appropriately perform the clearance against dynamic change for ontologies.

In practice, it is obvious that the improvement of performance should be reflective for the aim of applications and the characteristics of problems. It should be very engineering solution.

4.5 Related Work

So far, the efficiency of implementation of SWCLOS is compared with some reasoners which were reported in the LUBM paper by Guo et al. [22]. After the first LUBM reports at ISWC 2004, the LUBM Test has become the *de fact* standard for testing Semantic Web reasoners, and succeeding studies in Semantic Web community were carried out in two directions, that is, developing RDF stores for huge RDF or OWL Lite level ontology repositories, and the extending the LUBM test set to OWL DL or more comprehensive test suites. Virtuoso[8] and AllegroGraph[9] are examples of the former, and the efforts by Ma, et al. [45] and Weithöner et al. [76] are for the latter. In this section, the impact of early LUBM Test and the related work after LUBM are discussed.

4.5.1 Supplementary LUBM Test Reports

In the LUBM paper by Guo et al. [22], three reasoners, that is, DLDB-OWL, Sesami, and OWL-JessKB, were tested and other reasoners or tools were not adopted by various reasons. Then, supplementary reports were made for RacerPro, Pellet, and KAON2 by developer themselves of those tools.

RacerPro report

At the time of Guo's report, Racer failed to load the LUBM test dataset due to the inferior performance of connection to ABox. After that, the RacerPro was updated for the improvement of handling large ABoxes and the result was published on the HomePage[10] with the new trial version of RacerPro and data sets. It reports the time for loading, ABox preparation, consistency checking,

[8] http://virtuoso.openlinksw.com/
[9] http://www.franz.com/agraph/allegrograph/
[10] http://www.sts.tu-harburg.de/~r.f.moeller/racer/lubm.html

and index structuring is 13.87 + 1.38 + 0.00 + 7.87 sec for LUBM(1,0). The times for answering queries were varied from millisecond to a few seconds. It took 2.39 sec for Query 8.

Pellet report

Pellet is an OWL DL level reasoner in open source license. In Pellet, the tableaux-based reasoner searches for interpretation models in order to construct a completion graph of tableaux starting from ABox. The query form is strictly restricted within first order logic. However, owl:differentFrom is effective as well as rdf:type and owl:sameIndividualAs. Pellet can accept very restricted OWL Full descriptions that are accidentally described by users due to incomplete knowledge or careless miss in spite of aiming OWL DL, if they are “DLizable”. Subproperties of rdf:type and cardinality restrictions on transitive properties are ignored by Pellet.

Pellet reports results of LUBM(1,0), LUBM(3,0), and LUBM(5,0) [63]. It seems that Pellet showed better performance than RacerPro.

KAON2 report

KAON2 seems to be tested for LUBM(1,0), LUBM(2,0), LUBM(3,0), and LUBM(4,0) together with other ontologies in report [54]. However, the performance data of queries are disclosed only on three queries, that is, Query1, 2 and 3, although the report states, “As our results show, LUBM does not pose significant problems for KAON2."

4.5.2 Towards Complete Benchmark Suits

There are several pitfalls with respect to Benchmark Testing. Firstly, although benchmarking is basically useful to find out points of remedy on reasoners, excessive adaptation easily happens in trying to obtain a good performance with respect to materialized benchmark tests, because it is easy to adapt their systems to concrete test sets. It may be a problem if benchmark test sets are not matured and does not cover full range of problems, or in the case that the limitation of applicability stays ambiguous.

Secondly, creating benchmark test sets for ABox scalability is rather easy, but creating ones for TBox scalability is difficult. It should be noted that the complexity of TBoxes rises from the complexity of ontologies themselves. Thus, how to generate test sets for ontological complexity is left as a question to be solved.

Thirdly, we does not share common image of standardized OWL ontologies. There are many expressions to represent an ontology. People from OWL DL tend to use owl:equivalentClass together with owl:intersectonOf or owl:unionOf, although it is not necessary by OWL syntax rules. Some ontology has deeply nested RDF/XML expressions in which new entities appear only in the middle of a tree, and another ontology has a set of many shallow RDF/XML trees. Furthermore, ontologies seldom perfect on their syntax and semantics.

UOBM test sets

Ma et al. [45] pointed that LUBM does not provide test data on cardinality and allValuesFrom, and also pointed that it supports only a subset of OWL Lite. In fact, LUBM only covers 10/23 parts with respect to OWL Lite constructs. Aiming a complete benchmark test sets for OWL DL, Ma et al. extended LUBM to University Ontology Benchmark (UOBM).

In Ma's report [45], three reasoners, that is, OWLIM, DLDB-OWL, and Minerva were tested for UOBM test data. OWLIM[11] is a newly developed repository as a Storage and Interface Layer for Sesami. Minerva[12] is an ontology toolkit for a storage and the inference performed by Pellet at backend.

They also pointed out that LUBM has another limitation such that instances in ABox form multiple isolated graphs and the graphs lack links among them. This point was not important so far, but it has been becoming critical with recently spreading expectations for Linked Data.

ABox complexity vs. TBox complexity

Weithöner et al. [76] investigated the effects of the scalability of ABox onto the complexity of TBox. They claimed, "We are convinced that an ABox benchmark cannot be conducted without scaling the TBox size too. Inevitably this will also increase TBox reasoning complexity which again might influence ABox reasoning performance." Aiming to evaluate the scalability linkage between ABox and TBox, they created a new benchmark, and found the different behavior between RacerPro and KAON2 with respect to the size change of both TBox and ABox. However, it seems that scalability and complexity of TBoxes are vague.

[11] http://www.ontotext.com/owlim/
[12] http://www.alphaworks.ibm.com/tech/semanticstk/

Probabilistic ontology generation

To reduce required time and labor for generation of benchmarking datasets, the LUBM team studied how to rapidly generate the datasets, and generated Lehigh BibTeX Benchmark [75]. This direction of study seems to be important in order to produce complex TBoxes rather than ABoxes. However, there is no ontological consideration in Wang, et al. [75].

4.6 Concluding Remarks

The efficiency of SWCLOS implementation is tested by LUBM Benchmark Test Sets from hundreds thousand to one million triples. As a result, we conclude the following remarks.

1. SWCLOS showed the comparative performance in loading time and reasoning time to tools reported in Guo et al.
2. SWCLOS replied correctly to all queries of LUBM, whereas no reasoners but OWLJessKB in the reports could reply.
3. Refactoring by pursuing data dependency in triples was effective for complex query forms.
4. Backpointers are effective for complex query forms in order to produce effective code by pursuing data dependency.
5. Function memoization is also effective for efficient execution of reasoning and complex query forms.
6. We recognized that loading for LUBM takes long time in comparison of answering queries. There is a room of improvement for large-scale ABox loading. Persistentization of ABox and late evaluation for instances will be recommended.

It is obvious that ABox should be persistentized at least for a repository of ontologies. This result for LUBM Benchmark Test and discussion of related work indicated useful suggestions for building repositories, whereas SWCLOS is a language for Semantic Web application.

[This page intentionally left blank]

Chapter 5

Demonstration of OWL Full Metamodeling

"Demo or Die" (The MIT Media Lab's motto)

SWCLOS is the first full-fledged language system as OWL Full processor, in which the capability of metamodeling objects is borrowed from the power of the dynamic and reflective features of Lisp and metamodeling capability of CLOS. We implemented many OWL axioms into CLOS using Meta-Object Protocol (MOP) of CLOS.

Although unrestricted freedom of metamodeling certainly results in undecidability, most examples demonstrated as OWL Full undecidability are unreasonably extreme and make no sense from the view of engineering. In this chapter, several metamodeling examples of SWCLOS are shown within the understandable rationale of metamodeling from our practical experience, and a set of metamodeling criteria that enables SWCLOS to perform ontology metamodeling is addressed.

5.1 Tractability on Metamodeling and Metamodeling Criteria

5.1.1 Untractable Metamodeling

In SWCLOS, RDF(S) semantics is preserved in the OWL universe. Thus, it is critical to distinguish individuals, strict classes, and metaclasses according to the metamodeling manner of RDFS[1]. However, there some ontologies do not distinguish them such as shown in SUMO[2], and include direct

[1] See Section 6.3.

[2] `http://www.ontologyportal.org/`

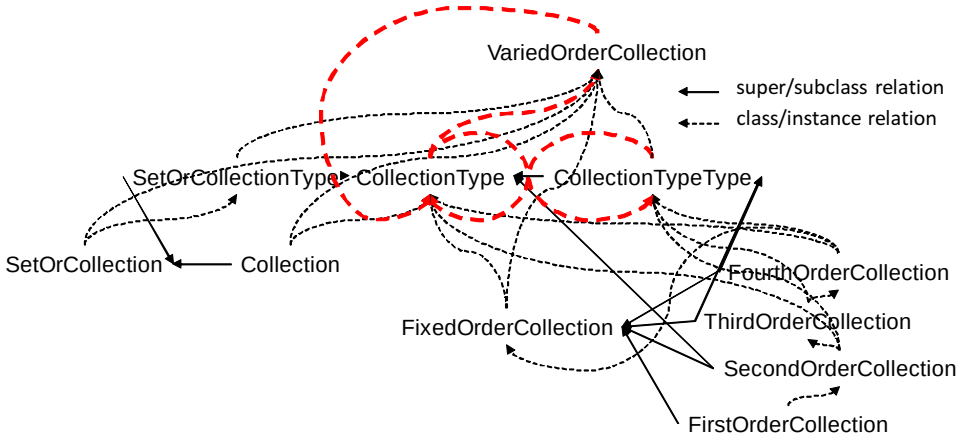

Fig. 5.1: Membership Loop in Cyc on Collections

and indirect infinte loops on membership such as shown in OpenCyc[3]. For example, as shown in **Fig. 5.1**, OpenCyc includes direct and indirect membership loops around the definitions of various kinds of collections (see the bold and broken curve lines). Such complex membership loops violate the metamodeling criteria which conform to the semantics of RDF(S) and SWCLOS.

In SUMO, although `sumo:UnitOfMeasure` and `sumo:SystemeInternationalUnit` are classes of class `sumo:Meter`, it does not satisfy the metaclass condition 6.16 (see Chapter 6). **Fig. 5.2** depicts this improper relation.

According to the semantics of RDFS and CLOS, SWCLOS does not process such unprincipled metamodeling. Thus, we addressed metamodeling criteria that conform to RDF(S) semantics and allow SWCLOS to perform metamodeling.

5.1.2 Metamodeling Criteria from RDF(S) Semantics

We settled criteria for ontology metamodeling that support the well-formedness of metamodeling with respect to the semantics of RDFS and SWCLOS. It is called *CLOS clean* or *RDF clean* metamodeling criteria. **Fig. 5.3** depicts some examples of CLOS clean metamodeling.

1. Ontology must be clearly layered in the base-object layer, the strict-class layer, and the metaclass layer. Every entity must be in only one layer and does not belong to more than one layer of them. In other words, it must be decidable for every entity in the universe which layer an entity belongs.

[3] `http://www.opencyc.org/`

5.

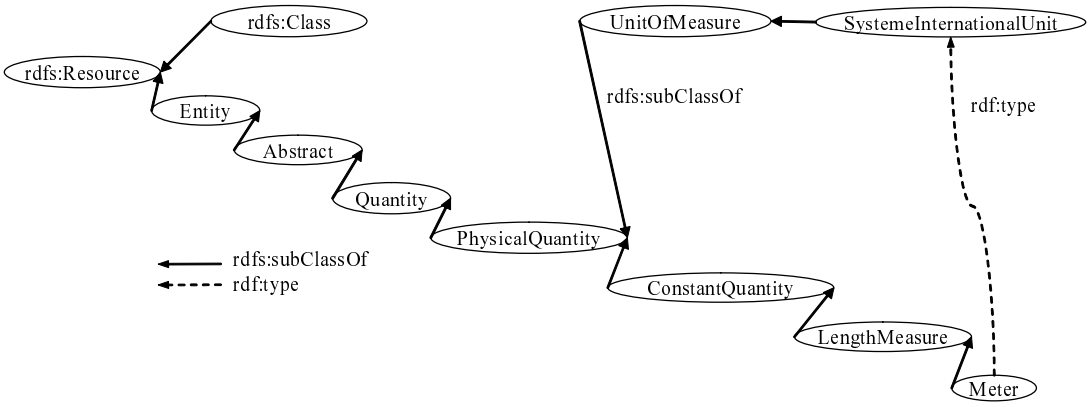

Fig. 5.2: A Part of SUMO Ontology

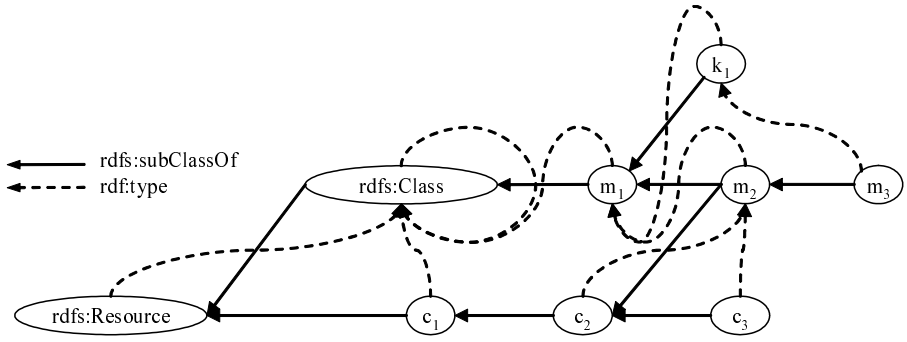

Fig. 5.3: Examples of CLOS Clean Metamodeling

2. There is no direct/indirect loop among superclass-subclass relation.

3. There is no direct/indirect membership loop except rdfs:Class.[4]

4. There may exist anywhere parallel relationship between superclass-subclass relation and membership relation. See m_1 and m_2, and m_1 and k_1 in Fig. 5.3. It may be called *stratified*.[5] This parallel relation bears a metaclassing bifurcation in metaclass layers but it cannot make a closed universe in ontology.

5. There may exist anywhere twisted relation between class super-subclass relation and membership relation. See c_2 and m_2 in Fig. 5.3. Such twisted relation makes a closed sub universe in the super universe.

6. A class of classes must have rdfs:Class in its superclasses. (metaclass condition)

Thus, in order to remedy SUMO ontology from the viewpoint of CLOS clean criteria, `sumo:Systeme-InternationalUnit` itself or `sumo:UnitOfMeasure` must be a subclass of rdfs:Class.

5.2 Demonstration of Metamodeling Programming

5.2.1 Treating a Class as Individual

Several usecases for metamodeling ontology are shown by W3C OWL Working Group.[6] All of them are easily programmed by SWCLOS. For example, for a class `a:Eagle` and an individual `a:Harry` defined as follows;

```
(/. a::Eagle rdf:type owl:Class)
(/. a::Harry rdf:type a::Eagle)
```

in order to define a class `a:Eagle` as instance of `a:EndangeredSpecies`,

```
(/. a:Eagle rdf:type a::Species)
(/. a:Eagle rdf:type a::EndangeredSpecies)
```

[4]Despite this criterion, we can set up substantially membership loops with subsumption using *twisted relation* or *proxy*. A twisted relation makes a subdomain that is completely included by a superdomain by making it between a universal class and a universal metaclass for the subdomain.

[5]In stratification of New Foundations (NF) set theory, for a given any formula $\varphi(x^n)$, the formula $\exists A^{n+1}\forall x^n[(x^n \in A^{n+1}) \Leftrightarrow \varphi(x^n)]$ is an axiom where A^{n+1} represents the set such that $\{x^n \mid \varphi(x^n)\}^{n+1}$. If the order of m_2 and m_3 in Fig. 5.3 is two, then the order of m_1 and k_1 is three.

[6]`http://www.w3.org/2007/OWL/wiki/Punning`

making `a::Species` a metaclass enables such metamodeling as follows.

```
(/. a:Species rdfs:subClassOf owl:Class)
(/. a:EndangeredSpecies rdfs:subClassOf a:Species)
```

5.2.2 Adding a Role Filler to a Class

Suppose the case such that wine brands must identify through ID-numbers by *International Wine Society*. Since there are brand wine concepts such as vin:Zinfandel mixed together with non-brand wine concepts such as vin:CaliforniaWine in Wine Ontology, we must distinguish them at first. Even if we introduce two new classes as a subclass of vin:Wine, namely BrandWine of which instances have an ID-number and NonBrandWine that does not have an ID-number, we cannot attach an ID-number to wine concepts such as vin:Zinfandel (in such case an ID-number is attached to wine instances such as vin:ElyseZinfandel), because in order to attach a role and filler to a class, a class of the class is required. The solution in SWCLOS is shown below.

```
(defConcept BrandWine (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class))    -> #<owl:Class BrandWine>
(defConcept NonBrandWineConcept (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class))    -> #<owl:Class NonBrandWineConcept>
(defProperty hasIDNumber (rdf:type owl:ObjectProperty)
  (rdfs:domain BrandWine)
  (rdfs:range xsd:positiveInteger))        -> #<owl:ObjectProperty hasIDNumber>
(defResource vin:Zinfandel (rdf:type BrandWine)
  (hasIDNumber 12345))                     -> #<BrandWine vin:Zinfandel>

(get-form vin:Zinfandel)
 -> (BrandWine vin:Zinfandel (rdf:about #<uri http://www.w3.org/TR ...
  (rdfs:subClassOf (owl:hasValueRestriction ...
                       ...
  (owl:intersectionOf vin:Wine
    (owl:hasValueRestriction ...
    (owl:cardinalityRestriction ...
  (hasIDNumber 12345))
```

5.2.3 Treating an Individual as Class

In Semantic Web Service by OWL (OWL-S) [47][7], the range of property process:hasPrecondition is expr:Condition, and an instance of expr:Condition may have a value of expr:expressionBody. Ordinary logics do not have a notion of class for logical expression, but suppose here that we have many kinds of conditions as definition and need to classify actual conditional occurrences to one of these condition classes. For example, we have many operational modes in the rocket launch operation [51], and each operational mode selects applicable services through preconditions. Note that an expr:expressionBody is different each other by operational modes and it identifies each condition class. Here we need to attach expr:expressionBody value to class-like preconditions, in order to classify actual conditions in operation and store them as instances of each operational condition classes. Recall that expr:expressionBody value may be attached to an instance of, but cannot be attached to, expr:Condition per se. In SWCLOS, the problem is solved as follows. We defined a new metaclass gxprcs:Precondition that inherits properties of expr:Condition, and then specific conditions, PipeCoolDownMode-, TankCoolDownMode-, and RocketTankingMode-Precondition were defined as instance of expr:Condition. As a result, actual occurrences are classified to three conditions, and similar but different services could be invoked according to the service precondition classes, within the boundary of the schema of OWL-S 1.1.

```
(defResource gxprocess::Precondition (rdf:type owl:Class)
  (rdfs:comment "This is a meta-class for precondition.")
  (rdfs:subClassOf owl:Class expr:Condition))
(defResource gxprocess::OperationModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "operation mode precondition")
  (rdfs:subClassOf expr:Condition gxdomain::OperationMode)
  (expr:expressionBody ... ))
(defResource gxprocess::PipeCoolDownModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "pipe cool-down mode precondition")
  (rdfs:subClassOf gxprocess::OperationModePrecondition
                   gxdomain::PipeCoolDownMode)
  (expr:expressionBody ... ))
(defResource gxprocess::TankCoolDownModePrecondition
```

[7] http://www.daml.org/services/owl-s/1.1/

```
    (rdf:type gxprocess::Precondition)
    (rdfs:label :en "tank cool-down mode precondition")
    (rdfs:subClassOf gxprocess::OperationModePrecondition
                     gxdomain::TankCoolDownMode)
    (expr:expressionBody ... ))
  (defResource gxprocess::RocketTankingModePrecondition
    (rdf:type gxprocess::Precondition)
    (rdfs:label :en "rocket tanking mode precondition")
    (rdfs:subClassOf gxprocess::OperationModePrecondition
                     gxdomain::RocketTankingMode)
    (expr:expressionBody ... ))
```

5.3 Concluding Remarks

In this chapter, firstly we saw examples for untractable metamodeling from the viewpoint of RDFS and CLOS metamodeling, and then the criteria for tractable metamodeling that is elaborated from the CLOS metamodeling capability and conforms to RDFS axioms[8] are addressed. According to the criteria of metamodeling, three typical metamodeling examples are demonstrated in SWCLOS.

The theoretical rationale of these metamodeling criteria and the foundation of OWL Full theory is discussed in the next chapter.

[8] See the next chapter.

[This page intentionally left blank]

Chapter 6

OWL Full Theory

> "*The most comprehensive formal systems that have been set up hitherto are the system of 'Principia mathemathica' (PM) on the one hand and the Zermelo-Fraenkel axiom system of set theory (further developed by J. von Neumann) on the other. These two systems are so comprehensive that in them all methods of proof today used in mathematics are formalized, that is, reduced to a few axioms and rules of inference. One might therefore conjecture that these axioms and rules of inference are sufficient to decide any mathematical question that can at all be formally expressed in these systems. It will be shown below that this is not the case, that on the contrary there are in the two systems mentioned relatively simple problems in the theory of integers that cannot be decided on the basis of the axioms.*" (On Formally Undecidable Propositions of Principia Mathematica and Related Systems I, Kurt Gödel)

In this chapter, OWL Full theory is developed with rearranging and rephrasing previously presented descriptions for RDF, RDFS, CLOS, and OWL in W3C documentations and Chapter 2.

Properly speaking, we do not believe that set theories are prerequisite to develop OWL Full Theory. However, futile arguments lasted about the foundation of semantic theory in Semantic Web community on the pretext of Russell's Paradox. The W3C recommendation of RDF semantics mentions Zermelo-Fraenkel set theory on one hand, and the other hand the W3C recommendation of OWL semantics mentions *comprehensive principle* that is a foundation of *naive* set theory by Georg Cantor. Set theories are the foundation of Semantic Web theory as well as they are the foundation of mathematical theory. Therefore, there is no choice but to discuss about set theories in order to rescue OWL Full from theoretic disorder, and then we discuss OWL Full theory.

6.1 Set Theory and Russell's Paradox

6.1.1 Comprehension Principle and Russell's Paradox

"In 1902, Ruessell wrote a letter to Frege, in which he informed Frege that he had discovered a paradox in Frege's *Begriffsschrift*. [...] Only six days later, Frege answered Russell that Russell's derivation of the paradox was incorrect. He explained that the self-application $f(f)$ is not possible in the *Begriffsschrift*. $f(x)$ is a function, which requires an *object* as an argument, and a function cannot be an object in the *Begriffsschrift*." (Kamareddine et al. [28] p.15)

A set can be extensionally defined by writing down all members of the set, or intensionally defined by designating a property that all members of the set satisfy. We usually use the latter to describe a set, because it is convenient to define an abstract set that has many or infinite number of members. For any given well-formed formula $\varphi(x)$, it is likely that a set such that satisfies the formula exists.

(Unrestricted comprehension) For an open well-formed formula φ,

$$\exists A \forall x[(x \in A) \Leftrightarrow \varphi(x)] \quad \text{where } \varphi(x) \text{ has } x \text{ free and has no free } A. \tag{6.1}$$

The unrestricted comprehension principle states that any formula $\varphi(x)$ such that A is not free in $\varphi(x)$ may be used for determining any set x. However, if we take $\varphi(x)$ to be $x \notin x$[1], which intends a set that does not have itself as a member, it causes a contradiction.

Suppose that $A = \{x \mid x \notin x\}$ exists. Then $\forall x[(x \in A) \Leftrightarrow x \notin x]$ from (6.1), therefore in particular A for x, $A \in A \Leftrightarrow A \notin A$. This is called Russell's Paradox, since a contradiction is derived from a system in which all of elements are plausible as individual.

There two workarounds followed Russell's Paradox. One, which is invented as Separation Principle by Ernst Zermelo, is to restrict the formula so that it does not involve the paradox, and the other is to introduce Russell's Ramified Type Theory.

6.1.2 Zermelo-Fraenkel Set Theory

In set theory, the uniqueness of sets are defined as extentionality principle:

[1] It is called *Russell's Class*. See Appendix A

Axiom 1 (Extensionality).

$$\forall A \forall B[\forall x(x \in A \Leftrightarrow x \in B) \Rightarrow A = B] \tag{6.2}$$

Zermelo weakened Cantor's Comprehension Principle (6.1) to his Separation Axiom.

Axiom 2 (Separation, Aussonderung). For an open well-formed formula φ,

$$\forall Z \exists A \forall x[(x \in A) \Leftrightarrow (x \in Z \wedge \varphi(x))] \quad \text{where } A \text{ does not occure in } \varphi(x). \tag{6.3}$$

Then, the set A in the Separation Axiom (6.3) is uniquely determined by Z and $\varphi(x)$ with (6.2). It is expressed in the usual notation as follows.

$$A = \{x \in Z \mid \varphi(x)\}$$

If $\varphi(x)$ is a property of sets such that a set A exists, then the members of A exactly satisfies φ. Namely,

$$x \in A \Leftrightarrow \varphi(x).$$

then this set A is denoted by $\{x \mid \varphi(x)\}$. Separation Axiom (6.3) separates a subset A from Z by formula $\varphi(x)$. Then, the member of set A satisfies a property $\varphi(x)$. It sounds like tautology and does not seem to be productive itself. Michael Potter [60] described on Separation Axiom that if Z is a set, then $\{x \in Z \mid \varphi(x)\}$ is a set. We do not have any useful information to construct A and Z from Separation Axiom itself, but it does not involve any paradox and useful as a template to support theorems and lemmas with other axioms.

In addition to the Separation Axiom, Axiom of Choice by Zermelo, Substitution Axiom by Abraham Fraenkel, and several other axioms have become a base of axiomatic set theory, and it is called Zermelo-Fraenkel Set Theory. See Appendix A.

Transfinite set theory

The set theory by Georg Cantor has been a mathematical foundation of the theory on natural number. Mathematicians have no interest in concepts of things in the world or ontology. They are interested only in mathematical concepts. They attempted to generate the concept of natural numbers by devising only sets of which members are also only sets.

Hereafter, we overview Zermelo-Fraenkel Set Theory according to the description by H.C.

Doets [15][2]. Let us start with axioms that (i) there exists at least one thing. and (ii) every thing is a set. We have already Extensionality Axiom (6.2) and Separation Axiom (6.3). Then, the empty set can be defined as follows.

Axiom 3 (Empty set).

$$\emptyset = \{x \in A \mid x \neq x\}$$

Note that this definition does not depend on the choice of A.

We define a successor operation S that takes a set and produces another set.

$$S(x) = x \cup \{x\}$$

These stipulations ensure to provide us the infinite number of sets.

$$\begin{aligned}
S(\emptyset) &= \emptyset \cup \{\emptyset\} = \{\emptyset\} \\
S(\{\emptyset\}) &= \{\emptyset\} \cup \{\{\emptyset\}\} = \{\emptyset, \{\emptyset\}\} \\
S(\{\emptyset, \{\emptyset\}\}) &= \{\emptyset, \{\emptyset\}\} \cup \{\{\emptyset, \{\emptyset\}\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \\
S(\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}) &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \cup \{\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} \\
\ldots &= \ldots \\
S(\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots) &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots\} \cup \{\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\}
\end{aligned}$$

Axiom 4 (Infinity). There exists a set A such that

$$\emptyset \in A$$
$$\forall x \in A[x \cup \{x\} \in A].$$

Notably such set A is closed to this axiom, and it has infinite members.

By making a map from the empty set to integer 0, and the successor S to a integer successor,

[2]The two chapters in [15] is summarized at Appendix A.

we obtain natural numbers due to two Peano systems with isomorphic.

$$\begin{aligned}
\emptyset &= 0 \\
S(\emptyset) &= \{\emptyset\} = 1 \\
S(\{\emptyset\}) &= \{\emptyset, \{\emptyset\}\} = 2 \\
S(\{\emptyset, \{\emptyset\}\}) &= \{\emptyset, \{\emptyset\}\} \cup \{\{\emptyset, \{\emptyset\}\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = 3 \\
S(\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}) &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} = 4 \\
\ldots &= \ldots
\end{aligned}$$

Theorem 1. *The natural numbers form a set.*

Thus, the concept of set of infinite number of natural number is obtained, Numbers that include ω is called *transfinite number* or *transfinite ordinal number*.

$$\omega = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\}$$

The successor operation can be applied to ω, then we obtain a series of transfinite numbers.

$$\begin{aligned}
S(\omega) &= \omega + 1 \\
S(\omega + 1) &= \omega + 2 \\
\ldots &= \ldots \\
S(\omega + \omega) &= \omega \cdot 2 \\
\ldots &= \ldots \\
S(\omega + \omega + \ldots) &= \omega \times \omega = \omega^2 \\
\ldots &= \ldots
\end{aligned}$$

Obviously, the magnitude of any transfinite number is bigger than the magnitude of any natural number, including a set of infinite members. Therefore, it is called *big*.

We strongly claim that we do not need to understand such *big* number to build any kind of ontologies that contain classes and instances of things in the real world. Namely, we do not need Zermelo-Fraenkel Set Theory and other axiomatic set theories developed for mathematical concepts, whereas the W3C Recommendation of RDF Semantics [25] states;

> When classes are introduced in RDFS, they may contain themselves. Such 'membership loops' might seem to violate the axiom of foundation, one of the axioms of standard (Zermelo-Fraenkel) set theory, which forbids infinitely descending chains of membership. However, the semantic model given here distinguishes [...] classes considered as objects from their extensions - [...] things that are 'in' the class - thereby allowing the extension of a [...] class to contain the [...] class itself without violating the axiom of foundation. (RDF Semantics [25])

Any set in Zermelo-Fraenkel Set Theory that satisfies the axiom of regularity[3] forbids the infinitely descending chains of membership. However, in proof it is far from the axiom of foundation to reach the forbidden chains of membership. Furthermore, it is not clear that the membership loop at the universal class does not violate the infinity of the descending chains of membership loop[4], whereas the recommendation states that the semantic model in RDF distinguishes classes as objects from their extensions. See (2.12). Rather we claim that it is appropriate to introduce the concept of type order in Ramified Type Theory in order to distinguish higher order rdfs:Class and lower order rdfs:Class for the sake of the prohibition on infinitely descending chains of membership on rdfs:Class. It is discussed later on.

Infinite descending chains of membership are also forbidden by KIF 3.0 Set Theory, and KIF Set Theory is appropriate to discuss Russell's Paradox in ontology, because the set is initially composed of ontological individuals.

6.1.3 KIF Set Theory

Knowledge Interchange Format (KIF) is a language designed for use in the interchange of knowledge among disparate computer systems. The universe of discourse in KIF is defined as the set of all objects presumed or hypothesized to exist in the world. If we start from a finite set of base entities that exist as individuals of a set but does not exists as a set, and construct a higher order class as a set of lower order objects, then we are never involved in Russell's Paradox, even if we have a class which has a member of itself. In fact, it is the way adopted in KIF [18], in which *bounded set* is distinguished from *unbounded set*[5].

The section on "Paradox" in KIF 3.0 Manual[6] states;

[3] See Appendix A

[4] Note that Zermelo-Fraenkel Set Theory does not include such a *universal class*.

[5] See also Appendix B.

[6] `http://logic.stanford.edu/kif/Hypertext/node25.html#SECTION0007400000000000000`

> "It is crucial that the paradoxes of set theory be avoided. One of the goals in the design of KIF is that it have a clearly specified model-theoretic semantics in terms of which the concepts of entailment, equivalence, consistency, soundness and completeness can be defined. If the paradoxes are allowed to persist in principle, even if they are easy to avoid in practice, the consequence would be that no KIF theory would be true in any model. Definitions couched in terms of models would be trivialized, becoming useless. All sentences would be entailed by any theory, any two theories would be equivalent, no theory would be consistent, every possible inference rule would be sound, and so on.
>
> In the von-Neuman-Godel-Bernays version of set theory, these paradoxes are avoided by replacing the principle of unrestricted set abstraction with the principle of restricted set abstraction given above. " (KIF 3.0 Manual)

An overview of such Set Theory is given in Appendix B. Whereas Russell's Paradox in ontology is forbidden in KIF Sets.

Russell's Ramified Type Theory is suitable to discuss metamodeling of ontology, because it includes the discussion about the order of types.

6.1.4 Ramified Type Theory

When Russell pointed out the inconsistency in his letter to Frege, he was just at the point of finishing his work. Then, he developed a type theory in order to work around the paradox which arose from unrestricted comprehension principle. His basic idea on type theory is to distinguish order of logical variables and predicates. If the order of arguments as variable in predicate calculus is zero, then the order of its predicate is one. If a variable as arguments in predicate calculus over first order objects in the domain, then the order of its predicate is two. Thus, it is critical to distinguish a variable nth order variable x^n from $n+1$th order variable x^{n+1}. A set of 0th order objects (individuals) is discriminated from a set of 1st order objects (predicates), and a set of nth order objects is discriminated from $n+1$th order objects.

Gilles Dowek [17] summarized the logic foundation of type theory as follows.

> To avoid Russell's paradox, and to get a (hopefully) consistent theory of sets, we can restrict naive set theory in two ways. The first method is to restrict the comprehension scheme to some particular propositions (for instance Zermelo's set theory permits four constructions : pairs, unions, power sets and subsets), the other is to move to a many

> sorted theory with a sort (called 0) for atoms a sort (called 1) for sets of atoms, a sort (called 2) for sets of sets of atoms, etc. and allow propositions of the form $t \in_n u$ only when t is of sort n and u of sort $n + 1$ (which permits to construct unions, power sets and subsets but disallows arbitrary pairs). The formalism obtained this way is called *higher-order logic* or *simple type theory*. The original formulation of A.N. Whitehead and B. Russell [...] has been modified by L. Chwistek, F. Ramsey and finally by A. Church [...].

Thus, it is obvious the substitution must be performed among the same order variables. In this theory self-referential typing does not occur within the discrimination among entities in different orders. See also Appendix C.

6.2 What is Comprehension Principle?

In the W3C recommendation of OWL semantics, the RDF-Compatible Model-Theoretic Semantics at Chapter 5 [58] describes the difference between the abstract syntax, which is developed in Chapter 3, and RDF compatible OWL DL as follows.

> "In OWL DL, localizing information must be provided for many of the URI references used. These localizing assumptions are all trivially true in OWL Full, and can also be ignored when one uses the OWL abstract syntax, which corresponds closely to OWL DL. But when writing OWL DL in triples, however, close attention must be paid to which elements of the vocabulary belong to which part of the OWL universe."(OWL Semantics Chapter 5 [58])

The 'localizing assumptions' are actually written in Chapter 5 of the W3C recommendation for blank nodes as instances of owl:Restriction for owl:onProperty restrictions, and blank nodes as instances of owl:Class for property owl:complementOf. Furthermore the notion of sequence for owl:oneOf, owl:intersectionOf, owl:unionOf, and owl:distinctMembers is introduced and the sequence is embodied as an instance of rdf:List. However, such 'localizing assumptions' are not required in OWL Full. For instance, we formalized owl:allValuesFrom restriction in formula (2.27). In RDF semantics, such a blank node for owl:allValuesFrom restriction actually can exist because of the entailment rule **rdfs4a** or the domain rule for owl:allValuesFrom.

For example, with an expression in the XML/RDF form for owl:Restriction such as;

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#x" />
  <owl:allValuesFrom rdf:resource="#w">
</owl:Restriction>.
```

the following is a demonstration in SWCLOS for such case.

```
gx-user(2): (addForm '(owl:Restriction (owl:onProperty x)
                                       (owl:allValuesFrom w)))
Warning: Range entailX3 by owl:onProperty: x rdf:type rdf:Property.
Warning: Range entailX3 by owl:allValuesFrom: w rdf:type rdfs:Class.
#<∀ x.w>
```

In the case of owl:oneOf, owl:intersectionOf, owl:unionOf, and owl:distinctMembers, we do not use range constraints of these properties, that is, rdf:List. Instead we directly embody the sequence as a list of Common Lisp as described in Subsection 2.2.1.

By the nature of RDF syntax and semantics, we can describe the same meanings in different ways using different RDF graphs. For example, we can describe vin:RedWine in Wine Ontology in the following two ways.
As one way with rdf:List;

```
vin:RedWine rdf:type owl:Class .
vin:RedWine owl:intersectionOf _:lst1 .
_:lst1 rdf:type rdf:List .
_:lst1 rdf:first vin:Wine .
_:lst1 rdf:rest _:lst2 .
_:lst2 rdf:type rdf:List .
_:lst2 rdf:first _:gx3  .
_:lst2 rdf:rest rdf:nil .
_:gx3 rdf:type owl:Restriction .
_:gx3 owl:onProperty vin:hasColor .
_:gx3 owl:hasValue vin:Red .
```

Then, as another way without rdf:List;

```
vin:RedWine rdf:type owl:Class .
vin:RedWine owl:intersectionOf vin:Wine .
vin:RedWine owl:intersectionOf _:gx4 .
_:gx4 rdf:type owl:Restriction .
```

```
_:gx4 owl:onProperty vin:hasColor .
_:gx4 owl:hasValue vin:Red .
```

The collection of objects that are indicated as a predicate value of a subject node can be simply represented a bunch of pairs of edges with an identical property name and objective graph nodes, without rdf:List. In comparison that each of container properties, i.e. rdf:Seq, rdf:Bag, rdf:Alt has clearly each semantics in their usage, respectively, there is no semantics in the usage of rdf:List and we have no motivation to use rdf:List in the usage for RDF graphs.

The document of RDF-Compatible Model-Theoretic Semantics [58] justifies the existence of a blank node using terminology 'comprehension principle'. However, this wording is misuse. As shown above, 'comprehension principle' means axiom (6.1) in set theories. On the other hand, 'comprehension principle' in OWL Semantics seems to mean a principle that requires the existence of individuals in several entailment rules[7]. We can find the same misuse on the terminology 'list comprehension' in some programming languages[8]. However, the function of the 'list comprehension' in those programming languages should be called 'separation principle', because the function is exactly the same as the meanings of 'separation principle' in ZF. To make matters worse, the word 'comprehension principle' is used on the pretext that RDF semantics involves Russell's Paradox. As described previous sections, Zermelo-Fraenkel Set Theory adopted the Separation Principle instead of comprehension principle in order to avoid the paradox. In KIF ontological Set Theory, Russell's Paradox is forbidden by the settlement of bounded sets (see Appendix B), and Russell himself developed Russell's Ramified Type Theory in order to avoid the paradox (see Appendix C).

Even if the universal class and the universal metaclass have a membership loop, we can get rid of the possibility of the infinitely descending chains of membership by distinguishing the orders of the universal class in descending process. We distinguish $rdfs:Class^{\mathcal{I}n}$ and $rdfs:Class^{\mathcal{I}n-1}$. The descending of orders never happens from $rdfs:Class^{\mathcal{I}n}$ to $rdfs:Class^{\mathcal{I}n}$ or $rdfs:Class^{\mathcal{I}n+1}$. The ontology metamodeling must be performed so that the descending of orders always happens from $rdfs:Class^{\mathcal{I}n}$ to $rdfs:Class^{\mathcal{I}n-1}$.

$$rdfs:Class^{\mathcal{I}n-1} \in CEXT^{\mathcal{I}}(rdfs:Class^{\mathcal{I}n}) \Leftrightarrow \langle rdfs:Class^{\mathcal{I}n-1}, rdfs:Class^{\mathcal{I}n}\rangle \in EXT^{\mathcal{I}}(rdf:type^{\mathcal{I}}) \tag{6.4}$$

[7] It is the same in OWL 2. See http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/#Appendix:_Comprehension_Conditions_.28Informative.29.

[8] http://en.wikipedia.org/wiki/List_comprehension

Therefore, if we start from a finite set of base entities that exist as individuals of a set but does not exists as a set, and construct a higher order class as a set of lower order objects, then we are never involved in Russell's Paradox, even if we have a class which has a member of its extension.

6.3 OWL Full Metaclassing

In Chapter 2, we derived the membership loop at the universal class and the universal metaclass from the characteristics of them. Namely, the universe class stands for the universe of discourse and the universal metaclass denotes all of classes in the universe of discourse. In this subsection, we show that their membership loops are derived from RDF axioms and consistent with axioms in RDF, by using the twisted relation between the universal class and the universal metaclass. Furthermore, we derive the metaclass condition.

6.3.1 Membership Loop at rdfs:Class and Twisted Relation with rdfs:Resource

Definition 1 (Class membership).

$$x^{\mathcal{I}} \in CEXT^{\mathcal{I}}(y^{\mathcal{I}}) \equiv \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf:type^{\mathcal{I}}) \tag{6.5}$$

Definition 2 (Universal metaclass).

$$\boldsymbol{C}^{\mathcal{I}} \equiv CEXT^{\mathcal{I}}(rdfs:Class^{\mathcal{I}}) \tag{6.6}$$

Definition 3 (Universal class, Twist1).

$$rdfs:Resource^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{6.7}$$

Definition 4 (Twist2).

$$\langle rdfs:Class^{\mathcal{I}}, rdfs:Resource^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \tag{6.8}$$

Axiom 1 (Subsumption). See also (2.13).

$$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \Rightarrow x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x^{\mathcal{I}}) \subseteq CEXT^{\mathcal{I}}(y^{\mathcal{I}}) \tag{6.9}$$

Axiom 2 (Reflection). See also (2.14).

$$x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x^{\mathcal{I}}, x^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \tag{6.10}$$

Axiom 3 (Top). See also (2.16).

$$x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x^{\mathcal{I}}, rdfs{:}Resource^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf^{\mathcal{I}}) \tag{6.11}$$

Axiom 4 (Meta).

$$x^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x^{\mathcal{I}}, rdfs{:}Class^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdf{:}type^{\mathcal{I}}) \tag{6.12}$$

Lemma 1 (Membership loop at universal metaclass).

$$rdfs{:}Class^{\mathcal{I}} \in \boldsymbol{C} \equiv CEXT^{\mathcal{I}}(rdfs{:}Class^{\mathcal{I}}) \tag{6.13}$$

Proof. By the definition of (6.8) and the subsumption axiom (6.9), we obtain $rdfs{:}Class^{\mathcal{I}} \in \boldsymbol{C}$.

Lemma 2 (Membership loop at universal class).

$$rdfs{:}Resource^{\mathcal{I}} \in CEXT^{\mathcal{I}}(rdfs{:}Resouce^{\mathcal{I}}) \tag{6.14}$$

Proof. By the definition of the universal class (6.7) and the subsumption axiom (6.9), we obtain $rdfs{:}Resource^{\mathcal{I}} \in CEXT^{\mathcal{I}}(rdfs{:}Resouce^{\mathcal{I}})$.

Making a conjunction of (6.7) and (6.8), we obtained the following.

$$rdfs{:}Resource^{\mathcal{I}} \in CEXT^{\mathcal{I}}(rdfs{:}Class)^{\mathcal{I}} \ \wedge CEXT^{\mathcal{I}}(rdfs{:}Class)^{\mathcal{I}} \subseteq CEXT^{\mathcal{I}}(rdfs{:}Resource)^{\mathcal{I}} \tag{6.15}$$

We call it *twisted relation* between rdfs:Resource and rdfs:Class. Note that this is resulted by the definitions of rdfs:Class and rdfs:Resource, in which the semantics of the universal class and the universal metaclass is applied to themselves.

Here, if we substitute $y^{\mathcal{I}}$ in Subsumption (6.9) by $rdfs{:}Class^{\mathcal{I}}$, then we obtained the following.

$$\langle x, rdfs{:}Class^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(rdfs{:}subClassOf) \Rightarrow x \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x) \subseteq \boldsymbol{C}^{\mathcal{I}} \tag{6.16}$$

This means a class that is a subclass of rdfs:Class is an instance of rdfs:Class and its class extension is also a subclass of the set of classes in the universe. Namely, this semantic condition is the same as that in CLOS semantics. In CLOS, it is very critical to distinguish base objects (individuals), strict classes (first order classes), and metaclasses (higher order classes). Then, we distinguish them by conditions;

1. $x \notin \boldsymbol{C}^{\mathcal{I}}$ (for base objects),
2. $x \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x) \not\subseteq \boldsymbol{C}^{\mathcal{I}}$ (for first order classes), and
3. $x \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x) \subseteq \boldsymbol{C}^{\mathcal{I}}$ (for metaclasses).

We call them, respectively, *base object condition*, *strict class condition*, and *metaclass condition*.

6.3.2 OWL Metaclassing

This subsection contains the re-arrangement of contents in Subsection 2.2.1.

Definition 5.

$$owl{:}Class^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{6.17}$$

$$\boldsymbol{OC}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Class^{\mathcal{I}}) \subset \boldsymbol{C} \tag{6.18}$$

$$owl{:}Thing^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \tag{6.19}$$

$$\boldsymbol{OT}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Thing^{\mathcal{I}}) \subset \boldsymbol{R} \tag{6.20}$$

$$owl{:}Restriction^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{6.21}$$

$$\boldsymbol{OR}^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}Restriction^{\mathcal{I}}) \subset \boldsymbol{OC} \tag{6.22}$$

$$owl{:}ObjectProperty^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \tag{6.23}$$

$$\boldsymbol{OP}_I^{\mathcal{I}} = CEXT^{\mathcal{I}}(owl{:}ObjectProperty^{\mathcal{I}}) \subset \boldsymbol{P} \tag{6.24}$$

The following is rephrasing of (3.30).

Axiom 5.

$$\boldsymbol{OC}^{\mathcal{I}} \subset \boldsymbol{OT}^{\mathcal{I}} \tag{6.25}$$

From (6.17) and (6.18), we see that owl:Class is a metaclass in RDF universe. From (6.21) and (6.22), we see that owl:Restriction is also a metaclass in RDF universe.

Furthermore, we obtain the following conjunction from (6.19) and (6.25)

$$owl{:}Thing^{\mathcal{I}} \in \boldsymbol{OC}^{\mathcal{I}} \ \wedge \ \boldsymbol{OC}^{\mathcal{I}} \subseteq \boldsymbol{OT}^{\mathcal{I}} \tag{6.26}$$

This is the same twisted relation as rdfs:Resource and rdfs:Class in RDF universe, so we can call it *twisted relation for OWL*. **Fig.6.1** shows the relationship between the two twisted relation. The

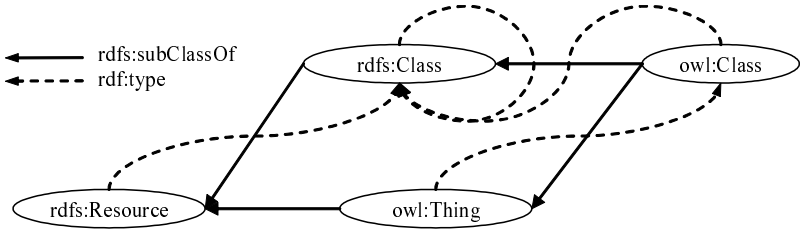

Fig. 6.1: RDF Universe and OWL Universe Connection

axiom (6.25) is required in order to let OWL classes be in the OWL universe. Namely, such twisted relation of (6.15) and (6.26) is effective so as to make a closed universe including classes. Notably, by connecting two universes of RDF and OWL such as depicted in Fig.6.1, OWL universe is constructed as a subset of RDF universe, and then OWL universe can be facilitated not only in OWL semantics but also in RDF semantics. Note that owl:Class and owl:Restriction are not in OWL universe rather in RDF universe.

6.4 Non-Unique Name Assumption and Equality

6.4.1 Equality of Individuals

Unique Name Assumption (by which different names always denote different entities), which is usually adopted into computer languages, is not adopted in Semantic Webs. In RDF, different URI references denote different graph nodes. However, in OWL language, owl:sameAs property may be applied to different URIs to indicate that two different URI references denote the same entity as individual in the OWL universe. Oppositely, the owl:differentFrom property (and the combination of owl:AllDifferent and owl:distinctMembers, too) may be used to indicate two different URI references denote different entities in OWL universe.

The followings are the same as the description in Subsection 2.2.1.
(owl:sameAs) If *x* and *y* is different URIs and the both references makes a pair that is an extension

of $owl:sameAs^{\mathcal{I}}$, then the denotation of x and y are the same one.

$$\{x \neq y \mid x \in \mathcal{V}_I,\ y \in \mathcal{V}_I\} \wedge \langle x^{\mathcal{I}},\ y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(owl:sameAs^{\mathcal{I}}) \Rightarrow x^{\mathcal{I}} = y^{\mathcal{I}}$$

(owl:differentFrom) If x and y is different URIs and the both references makes a pair that is an extension of $owl:differentFrom^{\mathcal{I}}$, then the denotation of x and y are different.

$$\{x \neq y \mid x \in \mathcal{V}_I,\ y \in \mathcal{V}_I\} \wedge \langle x^{\mathcal{I}},\ y^{\mathcal{I}} \rangle \in EXT^{\mathcal{I}}(owl:differentFrom^{\mathcal{I}}) \Rightarrow x^{\mathcal{I}} \neq y^{\mathcal{I}}$$

Thus, in case of no information on the equality in OWL, the equality of two entities is not determined[9], then, the decision of the equality of entity must be performed in the RDF universe. To discuss the equality of entities in RDF semantics, it is appropriate to discuss the equality of two subgraphs that the two entities are in position of *subject*.

The algorithm for the equality computation in the RDF universe is explained as follows[10].

Two RDF graphs G and G' are equivalent if there is a bijection $\mathcal{M}$ between the sets of triples for the two graphs, such that:

1. $\mathcal{M}$ maps blank nodes to blank nodes.
2. $\mathcal{M}(lit) = lit$ for all RDF literals *lit* which are nodes of G.
3. $\mathcal{M}(uri) = uri$ for all RDF URI references *uri* which are nodes of G.
4. The triple $s/p/o$ is in G if and only if the triple $\mathcal{M}(s)/p/\mathcal{M}(o)$ is in G'.

Note that these are not described in denotational semantics. The document of RDF Semantics [25] state;

> Any instance of a graph in which a blank node is mapped to a new blank node not in the original graph is an instance of the original and also has it as an instance, and this process can be iterated so that any 1:1 mapping between blank nodes defines an instance of a graph which has the original graph as an instance. Two such graphs, each an instance of the other but neither a proper instance, which differ only in the identity of their blank nodes, are considered to be equivalent.

[9] Instance properties of owl:FunctionalProperty and owl:InverseFunctionalProperty also affect the equality as individual.

[10] `http://www.w3.org/TR/rdf-concepts/#section-graph-equality`

For the discussion of equality under the non-UNA condition, we superimpose owl:sameAs and owl:differentFrom properties onto the above algorithm. RDF is property-centric but OWL is object-centric (it means a subject node and linked nodes with one hop predicates are regarded as an object like object-oriented language). Then, we modify the above algorithm to meet OWL object-centric paradigm. For each subgraph composed of a subject node and one-hop linked nodes in G and G',

1. $\mathcal{M}$ maps blank nodes to blank nodes.
2. $\mathcal{M}(lit) = lit$ for all RDF literals *lit* which are nodes of G.
3. $\mathcal{M}(uri^I) = uri^I$ for all RDF URI references *uri* which denote nodes of G.
4. For every s of triple $s/p/o$ for G, $\langle s^I, o^I \rangle \in EXT^I(p^I)$ if and only if $\langle \mathcal{M}(s^I), \mathcal{M}(o^I) \rangle \in EXT^I(\mathcal{M}(p^I))$ is in G'.

In case that s^I in G and s'^I in G' are blank nodes in a bijection $\mathcal{M}$, $\langle s^I, o^I \rangle \in EXT^I(p^I)$ is equivalent to $\langle \mathcal{M}(s^I), \mathcal{M}(o^I) \rangle \in EXT^I(\mathcal{M}(p^I))$, if $o^I = \mathcal{M}(o'^I)$ in OWL semantics. We apply the same algorithm for non-blank node in non-UNA condition. In case that s^I and s'^I are named with different names and we cannot determine the equality by the names, our approach determines the equality between s^I and s'^I through the subgraphs of the both. Namely, we check p^I and the equality of o^I and o'^I. This algorithm traverses two graphs, until the decision is obtained. Note that RDF graph is a directed graph. In this graph equality checking, if two nodes have sub-trees, the corresponding sub-trees on both graphs are recursively checked for the equality. Thus, if we reach at terminal nodes (atomic nodes that do not have edges any more) but no information is obtained, we fall into a troublesome situation. For example, in comparison of ex:Y/ex:p/ex:A and ex:Z/ex:p/ex:B, if ex:A and ex:B are both atomic, the non-UNA computation cannot conclude whether or not ex:Y is equivalent to ex:Z. In such condition, in order to derive useful computational results, we must define the equality or difference among every atomic individual. It is very laborious work to describe common knowledge such as Bill is different from George, Barack, Al, and so on.

Therefore, we devised a flag for non-UNA and set up falsity to the flag as default. Note that the equality of two blank nodes is checked both in UNA and in non-UNA. In the default condition, we stand in UNA as well as for ordinary computer languages, then two nodes that have different URI references are different, and then two blank node trees are distinct if we cannot find the corresponding edges of graphs or we find the lexically different URI references at the corresponding positions in the trees. In non-UNA condition with the flag setting, the graph equality checking is performed even though two URIs at the corresponding positions are different, until we find either the difference

of graph structures or the difference of nodes that are explicitly stated in OWL statements. In our approach, two atomic nodes with different names are regarded as different in the equality checking, even though the flag indicates non-UNA. Thus, this algorithm is paraphrased *UNA for atomic objects in the non-UNA condition*.

6.5 Concluding Remarks

In this chapter, we gave an overview of Zermelo-Fraenkel Set Theory, which is the standard set theory in the number theory of mathematics and does not involve Russell's Paradox in system. We strongly claimed that we do not need such *big* number set theory for ontology construction. The overview of KIF Set Theory is also given, and it is confirmed that Russell's Paradox is avoided by bounded sets in KIF. As the other way for avoiding Russell's Paradox, Ramified Type Theory by Whitehead and Russell is also mentioned and it is stressed that it is critical to discriminate an identical entity in syntax by its order in semantics. The position in which an entity appears with respect to membership (in set theory) and predicate or terms (in higher order logic) is critical. The problem of 'localizing assumptions' described in the RDF-Compatible Model-Theoretic Semantics in W3C Recommendation is discussed, and the misuse of word 'comprehension principle' is pointed out.

OWL Full metaclassing is re-formalized using by RDF(S) Axioms, and OWL universe is formalized as subset of RDF universe. The membership loop at the universal class rdfs:Resource and the universal metaclass rdfs:Class is also discussed using RDF(S) Axioms.

Finally, non-Unique Name Assumption in OWL is superimposed onto RDF graph, and the novel algorithm for Unique Name Assumption for atomic objects in the non-UNA condition is invented in order to integrate OWL non-Unique Name Assumption to RDF graph.

[This page intentionally left blank]

Chapter 7

Open World Assumption and Class Disjointness

> *"There are more things in heaven and earth, Horatio, than are dreamt of in your philosophy."* (Shakespeare, Hamlet)

Concerned with the existential restriction of property or owl:someValuesFrom, the full Open World Assumption (OWA) is meaningless from the viewpoint of ontology building, since the existential restriction under the OWA means the possibility that a satisfiable value may be defined somewhere in WWWs or someone in the team members may add a proper constraint tomorrow or after. The full OWA implies that ontology builders cannot know all for target ontologies. However, this assumption is not enjoyable in actual fact in personal and collaborative ontology building process. Thus, we settled more mild setting for the problem of Open World Assumption.

7.1 Auto Epistemic Closed World Assumption

It is natural to distinguish the local world for target ontologies and the given general WWW. Hence, we have introduced the notion of *auto-epistemic local closed world assumption*. In this idea, agents can introspectively check their knowledge within their extent of capabilities.

An agent sits in locally closed world as environments around it. The flag for auto-epistemic local closed world assumption is set true as default in SWCLOS, and the satisfiability for slot value is aggressively checked even in case of the existential restriction. Namely, if an existential restriction is not satisfied, then the interpretation is not satisfied. Setting the flag false means the completely

full OWA. In this case, no alarm is signaled for the existential restrictions.

7.2 Complete Relation for Class Equivalency and Disjoint Relation

In OWL, owl:equivalentClass is applicable to indicate the equivalency of two objects as class. For example, food:Wine in Food Ontology[1] is equivalent to vin:Wine in Wine Ontology[2] with the statement of owl:equivalentClass. In addition, the other three complete relations[3], i.e., owl:intersectionOf, owl:unionOf, and owl:oneOf also decide the equivalency of classes. In case that two concepts (classes) have equivalent values for these complete relational properties, the two concepts must be conceived to be equivalent. For example, vin:DryWine and vin:TableWine in Wine Ontology are equivalent as class in OWL semantics (they share the same class extensions), because the both have the same value for owl:intersectionOf property.

Meanwhile, we can actively apply the statement of owl:disjointWith to classes that we consider they are disjoint each other. In addition, owl:complementOf property explicitly state that two concepts are definitely different as class. Thus, in case of no declaration of equivalency and disjointness of classes, we cannot determine the equality as classes immediately.

However, the complete relations except owl:equivalentClass decide not only the equality but also the disjointness of classes. For example, even though we have no direct statement of disjointness for vin:RedWine and vin:WhiteWine, the disjoint relationship between them is deduced through property owl:intersectionOf and owl:hasValue restriction vin:Red of vin:RedWine and vin:White of vin:WhiteWine, because it is explicitly stated that vin:Red is different from vin:White. Furthermore, we can also conclude some useful results by resorting to RDF graph checking mentioned at the previous chapter. For example, we can find that vin:CaliforniaWine is not equal to vin:ItalianWine in spite of no explicit information of disjointness, because the graph equality checking deduces that vin:CaliforniaRegion, in which vin:CaliforniaWine is located, is different from in:ItalianRegion, in which vin:ItalianWine is located, even if we are in non-UNA.

However, for atomic concepts that have no edges except being pointed as superclass, we cannot conclude that Man is disjoint to Woman, if those concepts are atomic in non-UNA. Thus, we are forced to do very laborious work to describe common knowledge such as Man and Woman are disjoint, Plant and Animal are disjoint, Ape and Monkey are disjoint, Virus and Bacteria are disjoint,

[1] http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf
[2] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf
[3] http://www.w3.org/TR/owl-ref/#DescriptionAxiom

and so on[4].

ANSI Common Lisp specifies that CLOS classes are pairwise disjoint if they have no common subclass and one class is not a subclass of the other. Namely, each class is disjoint to the others as default until we connect them in superclass relation or set a common subclass. This agreement is supported by the premise that an object in CLOS is typed to only one class. In the RDF universe, an entity may be typed to more than one class. So, the nature of disjointness in CLOS is not applicable in the RDF universe in theory. However, in SWCLOS, the pseudo multiple-classing machinery[5] is implemented using the CLOS class and multiple inheritance mechanism. Therefore, from the viewpoint of CLOS, the algorithm of disjointness for CLOS is still valid in the RDF universe in virtue of CLOS. In the next section, we introduce an idea of role concept that is divided from substantial concept with the premise of pairwise disjointness.

As described in Chapter 2, the subsumption (2.13) in RDF semantics is weak but the subsumption (2.22) in OWL semantics is strong. Therefore, we must pay attention to determine class disjointness which semantics we are talking about as OWL Full Theory.

Note that if two classes are implicitly or explicitly stated as disjoint in OWL semantics. Then, both must be disjoint in RDF semantics. In such a case, the error must be signaled, if the non-disjoint relation is given later on. Similarly, in case that two classes are in the non-disjoint relation in RDF semantics, the error must be signaled, if the implicit or explicit disjoint relation is given later on. Oppositely, if we have some evidence of subsumption in OWL semantics, it must be interpreted not disjoint even if there is no relation in RDF semantics. Thus, if we have no evidence of disjointness and subsumption in OWL semantics and no evidence of subsumption in RDF semantics, we interpret it disjoint as default. However, this default disjointness accepts the change later on when new statements are inputted.

7.3 Pairwise Disjoint Datatype

In SWCLOS, we defined XML datatype wrappers as mapping XML Schema descriptions to lisp datatypes in lisp value space. We defined xsd datatypes in lisp space and the same datatypes are also defined as CLOS classes in the RDF universe in order to treat them in the RDF universe. Therefore, independent XML Schema datatypes in SWCLOS are pairwise disjoint by the implicit disjointness of CLOS classes, e.g., xsd:float is disjoint with xsd:integer, xsd:URI, xsd:string, xsd:boolean, etc.

[4] As mentioned in Chapter 1, 58% is for class disjointness in lines of pizza.owl for only 23 pizza and 29 pizza toppings. The number of lines for disjointness will explode with the number of classes.

[5] See Subsection 3.1.6.

7.4 Ontological Categories and Disjointness

OWL provided the description of class disjointness and forced us labor-intensive work as described above. W3C new recommendation for OWL, namely OWL 2 specification [9], attempts to solve the disjointness problems without ontological consideration in depth. In OWL 2, Person may be described as owl:disjointUnionOf Man and Woman. However, we are still forced to describe explicitly disjointness for all disjoint classes, or basic atomic concepts. We strongly claim that the approach to describe disjointness must be well-founded on ontological consideration.

Sowa [66, 67] showed a lattice of the top-level ontological categories of things. Each of the twelve elemental concepts in the top ontology has different characteristics and those combinations, i.e., *independent*, *physical*, *relative*, *abstract*, and *mediating*. The concepts of the independent exist itself and they show the firstness. The concepts of the relative or *role* only live with the firstness and they show the secondness. The mediating describes concepts that mediate the firstness and the secondness.

Guarino [20] parted ontology into two categories, i.e., *particular* that represents substantial entities and *universal* that is the category of entities required to describe the particulars. Physical objects, abstract processes, phenomena, quality, and materials fall into the particular, and attributes, relations are categorized into the universal.

Mizoguchi, et al., developed an ontology building tool called Hozo [53, 43] based on ontological deep discussion and have utilized Hozo for many application field of ontology building. Using Hozo, ontology builders can easily construct complex concepts that are composed of substantial sorts and non-substantial roles. For example, Wife is a part of Family and composed of Woman and Wife-role. The concept Woman is a substantial and may have slots of gender, age, etc. The role concept Wife-role is not a substantial, in other words, it always requires substantial concepts to work, but may have its own slots such as married-year, partner, etc. In a sense, it is regarded that the concept Family represents the context in which the concept Wife is activated from Woman with Wife-role.

Takeda, et al., also proposed *Aspect Theory* of ontology in the study of *Knowledgeable Community* [70], which is a framework of knowledge sharing and reuse based on a multi-agent architecture. In this framework, while ontologies are the minimum requirement for each agent to join the community, each of heterogeneous ontologies describes an aspect of an entity and knowledge. A mediator agent that embodied knowledge for mediation helps other agents to communicate each other. In this

theory, the aspect may be rephrased as a context on which an agent focused for discourse. For example, a concept Temple is an aggregation of concepts in aspect of religion, cultural asset, building architecture, corporate body, and so on. In most case without communication, we usually focus on one aspect of entity and do not need to take care of the other aspects in a particular context. However, for agents in a particular discourse, the mediator translates heterogeneous ontologies from one to another and mediates agent's speech acts that are broad-casted in the community.

7.5 Introduction of Role Concepts

In order to solve the labor-intensive disjointness problem, we propose two ontological categories according to Kozaki, et al. [43], i.e., substantial concepts and role concepts, and realize them on top of RDF and extended OWL semantics. The substantial concepts are described in OWL, but we adopt the assumption of implicit disjointness for substantial concepts in the same way as CLOS described above. On the other hand, a role concept is an extension of owl:Restriction. Neither owl:disjointUnionOf nor owl:AllDisjointClasses in OWL 2 are introduced. Instead, we extend owl:Restriction, which has property-value restrictions but usually no name and no super restrictions in OWL, to the role concept that is able to have a name and supers. The instance of role is attached to an instance of substantial classes in the same way as owl:Restriction provides the definition of *predicate/object* at *subject* or an instance of substantial class. A complex concept is composed of a substantial class and role concepts. For example, a complex concept Husband is composed of Man and Husband-Role that has spouse and marriage-date properties, and Teacher is composed of Person and Teacher-Role that has subject and classInCharge property. The discussion of disjointness on role concept is meaningless, because the role concept cannot have any instance by itself as well as owl:Restriction. Husband and Teacher can share individuals, but those individuals should be interpreted as instances of Man in Husband and Person in Teacher.

7.6 Concluding Remarks

In this chapter, the problem of full setting on Open World Assumption is pointed out, and more practical and useful setting of auto-epistemic local closed world assumption is introduced. If this flag is set, owl:somevaluesFrom restriction is aggressively checked against loaded ontologies into SWCLOS. In order to avoid wasteful alarming by forward referencing, the flag is set as false in batch loading mode, but it is set as default in interactive mode.

Secondly, the difference of semantics of disjointness in RDF semantics and OWL semantics is described. The algorithms for the disjointness described here are embodied in SWCLOS.

Thirdly, the problematic OWL disjointness is pointed and the novel approach is proposed under the consideration of ontological sorts of substantial (disjoint each other in nature) and non-substantial sort (role concepts). It is suggested that these two sorts can be realized as the extension of OWL 1 semantics.

Chapter 8

Related Work

8.1 Frame-based and Object-Oriented OWL Systems

In this doctoral study, an OWL Full system is realized by the object oriented system in Common Lisp language. However, this is not the first case of OWL realization of object oriented systems. Meditskos and Bassiliades presented a deductive object-oriented system O-DEVICE for reasoning over OWL [50]. This system is based on CLIPS Object-Oriented Language (COOL). This system intends to extend to Rule language based on CLIPS.

F-OWL is an ontology inference engine for the Web Ontology Language OWL by Harry Chen, Youyong Zou, Lalana Kagal and Tim Finin[1]. The ontology inference mechanism in F-OWL is implemented using Flora-2, an object-oriented knowledge base language and application development platform that translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine. The ability to support knowledge consistency checking using axiomatic rules defined in Flora-2. It is also stated that F-OWL is still in its early stage of development.

F-logic is a frame system of object oriented logic programming [33, 32], and Heiko Kattenstroth studied the combination of OWL DL and F-logic and realized the knowledge-management system using Jena and Florid (F-Logic Reasoning in Databases) [30] as a result of PhD thesis. However, it is based on OWL DL and not OWL Full. Therefore, there is no discussion for OWL Full theory in this study.

Recently, Puleston, et al. [61] made efforts to integrate Java language to OWL for the purpose of developing Semantic Web application in the domain of Health Care and the Life Science. However, it is very difficult in this approach to extend the system to OWL Full, because Java language is

[1] http://fowl.sourceforge.net/about.html

strictly limited to the functionality of reflection and no Meta-Object Protocol. We think it is hopeful that the UML framework by OMG will be extended to allow OWL Full metamodeling.

8.2 RDF and OWL Theory

8.2.1 RDF Semantic Theory

Bruijn et al. presented logical analyses of RDF reasoning and complexity [12, 13]. In these paper, they pointed out that the triple construct and class membership in RDF is close to the attribute value construct in F-Logic [32]. They also discussed blank nodes, domain constraints, and *class position* and *individual position* of term. It is concluded that complete RDF, which includes blank nodes and some of rdfs:Resource, rdfs:Class, rdf:Property, rdfs:ContainerMembershipProperty, and rdfs:Datatype, has the complexity of NP-complete.

The Close World and Open World in RDF semantics was discussed in Analyti et al. [3]. In this paper, they extended RDF graph to what allows explicit strong negation, and then created new terms $erdf:TotoalProperty$ and $erdf:TotoalClass$ that represent metaclasses, on which the Open World Assumtpion applies. They defined ERDF models and Herbrand interpretations. However, we consider that it is important to contribute the progress of OWL itself and OWL Full according to the original idea for Semantic Webs by Tim Berners-Lee.

8.2.2 OWL Semantic Theory

Since achieving OWL 1 of W3C Recommendation, the shortage of the OWL specifications has been widely recognized through many practical experience of applying OWL language, and then the effort of the improvement has resulted in new W3C recommendation of OWL 2. The affairs of OWL 2, motivations, conditions, objectives, etc. were reported in Grau et al. [19].

ODM team also tackled OWL DL and OWL Full for the integration to UML [8]. In this paper, they discussed the integration of OWL DL and OWL Full onto UML profile.

The efforts to computation of OWL Full is also continued by Jena team, and two important contributions are performed. Turner and Carroll [74] found many minor errors in the specification of the semantics of OWL, using interactive theorem prover *Isabelle*. Successively Carroll and Turner [10] discussed OWL Full theory and ‘comprehension principle’, and then concluded that OWL Full can be consistent without ‘comprehension principle’. However, it is wonder that these contributions seems to be not evaluated in Semantic Web community.

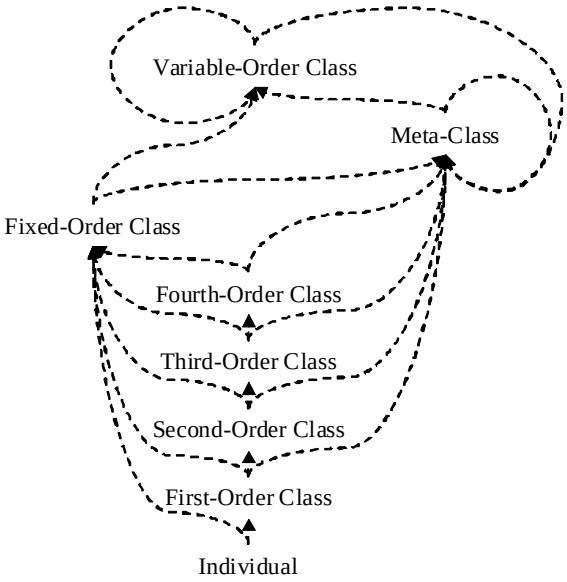

Fig. 8.1: Membership Loop in Cyc by foxvog [16]

8.2.3 Criteria for Metamodeling

We proposed the criteria for well-founded metamodeling in OWL Full. We believe this proposal is very critical to establish OWL Full, and in our best knowledge at this time there is no similar proposal. Foxfog [16] reported rules for sound membership loop in Cyc. The complex membership loops in Cyc are depicted in **Fig. 8.1** which is taken from [16]. However, the rationale of the Cyc membership loop and the membership rules is not reported.

8.3 Other Work

Recently, with the progression of research on Linked Open Data, the ambiguity of the semantics on owl:sameAs is coming to be closed up. Halpin et al. has pointed the problems on owl:sameAs with respect to Linked Data in [24]. The problematic Open World Assumption is also becoming known gradually. In SWCLOS, we simply settled the flag for auto-epistemic closed world assumption, Pellet also has equipped with a flag for CWA. Alferes, et al. [2] introduced the logic of MKNF (minimal knowledge and negation as failure) into query for the purpose of combining OWA and CWA. We think hopefully this direction is a legitimate direction as the solution in logic theory.

8.4 Concluding Remarks

In this section, related work is described from the survey on recent work with respect to the OWL system realization, especially focusing frame systems and object-oriented systems. The research results on OWL DL and Description Logics are omitted in this dissertation, but some system realizations in OWL DL are reported in the chapter of LUBM 4.

RDF and OWL theory also investigated and the contributions from HP Jena Team is admitted. We also found a few reflections on individual sameness and the problematic Open World Assumption, which are recently closed up as problems.

Chapter 9

Conclusion

In this doctoral study, OWL semantics is formalized in Tarskian denotational semantics as well as RDF Semantics according to the description of RDF Semantics [25] of W3C Recommendation, with focusing mainly membership and subsumption of entities. Due to no formalization so far on OWL semantics by Tarskian denotational semantics, this part is our original contribution for Semantic Webs. CLOS semantics and its computational model are also discussed based on the model addressed by Brian Smith, and the semantic gap between OWL and object oriented languages addressed by SETF of W3C are pointed out.

The semantics of RDF(S) is very close to that in CLOS. Then, the RDFS type structure is straightforwardly mapped onto the type system of CLOS. The semantic gap between object-centric CLOS and property-centric RDF(S) is filled up by setting property resource objects in CLOS and inventing the collection mechanism for the property extension in RDF(S) through CLOS native slot-definition facilities. The flexible implicit slot definition on demand is embodied in the class-based CLOS system. In order to accept forward-reference for entities, the novel functionality called *proactive entailment* is realized using RDF/OWL entailments. The domain and range constraint were developed with the property inheritance mechanism and embodied into SWCLOS.

In order to realize OWL universe in RDF universe, only one axiom that owl:Class have to be a subclass of owl:Thing was added into the set of OWL axioms, which are described in the OWL description file. All semantics and functionality of OWL specification was implemented on top of RDF(S) subsystem with preserving RDF(S) semantics.

The efficiency of SWCLOS implementation is tested by LUBM Benchmark Test Sets. As a result, SWCLOS showed the comparative performance in loading time and reasoning time to tools reported in Guo et al., and SWCLOS replied correctly to all queries, whereas no reasoners but

OWLJessKB in the reports could reply.

Several examples of metamodeling in OWL Full is presented with the criteria for tractable metamodeling that is elaborated from the CLOS metamodeling capability and confirms to RDFS axioms are addressed.

OWL Full theory is presented in metaclassing formalization based on RDF(S) Axioms, and integration of RDF universe and OWL universe is described based on this metaclassing theory. The non-Unique Name Assumption in OWL is superimposed onto RDF graph, and the novel algorithm for Unique Name Assumption for atomic objects in the non-UNA condition is invented in order to integrate OWL non-Unique Name Assumption to RDF graph. The mild settings for Open World Assumption is indicated as auto-epistemic closed world assumption, and the introduction of role concepts is suggested as the extension of the OWL specification in order to solve the problematic disjointness of OWL classes.

The overview of set theories on Zermelo-Fraenkel and KIF are given for the purpose of avoiding futile arguments on membership loop and Russell's Paradox.

As a result of this study, SWCLOS has become the first full-fledged language as OWL Full processor, in which the capability of metamodeling objects is borrowed from the power of the dynamic and reflective features of Lisp and metamodeling capability of CLOS.

We hopefully desire that the initial goal of Semantic Webs that indicated by the Semantic Web Layered Architecture will be achieved in the near future. Simultaneously, Common Logic[1] is emerged as alternative language that targets Semantic Web domain. Although the detail of the specification is still open and to be fixed as soon as possible, it is already established as an abstract language of ISO standard in a logic framework intended for information exchange and transmission. The framework allows a variety of different syntactic forms, called dialects, all expressible within a common XML-based syntax and all sharing a single semantics. The dialects, which have different syntax but interchangeable from one to another, include Common Logic Interchange Format (CLIF)[2], Conceptual Graph Interchange Format (CGIF)[3], XML Common Logic (XCL)[4], and Common Logic Controlled English (CLCE)[5]. CLIF may be conceived to be a modernized version of Knowledge Interchange Format (KIF)[6][26]. **Table 9.1** summarizes the basic computational assumptions underlying Common Lisp, Description Logic, OWL, and Common Logic.

[1]http://common-logic.org/
[2]http://www.ihmc.us/users/phayes/CLIF.html
[3]http://conceptualgraphs.org/
[4]http://www.altheim.com/specs/xcl/1.0/
[5]http://www.jfsowa.com/clce/specs.htm
[6]http://www-ksl.stanford.edu/knowledge-sharing/kif/

Table 9.1: Basic Computational Features of Languages.

	UNA/nonUNA	CWA/OWA	Truth Value	arity
Common Lisp	UNA	CWA	Ternary	n
Description Logic	UNA	(OWA)	Binery	2
RDF(S)	UNA	(CWA)	Binery	2
OWL	nonUNA	OWA	Binery	2
Common Logic	nonUNA	OWA	Ternary?	n

The abstract syntax model of Common Logic is analogous to polymorphism in object oriented programming and no fixed arity like Common Lisp (*polyadic*). As shown in Table 9.1, the arity of RDF and OWL is strictly constrained to 2. Not only Common Logic allows *n*-ary, but also the arity is not fixed for a predicate or property. Guha and Hayes initially proposed such features as the RDF syntax for a common base language of Semantic Web languages [21]. In their initial proposal for the candidate of RDF, they expected a base language for Semantic Web languages, and claimed the basic language, called L_{base}, that supports basic inference and semantics, and then allows RDF and extending different semantics at the upper layers in the Semantic Web stack. They imagined that L_{base} provides L_i language in *i*-th layer of Semantic Web language stack. However, the history of Semantic Web languages did not developed as Tim Berners-Lee intended.

Common Logic also permits 'higher-order' constructions such as quantification over classes or relations while preserving a first-order model theory. The semantics allows theories to describe intensional entities such as classes and properties. The first solution of this 'higher-order' constructions will be *metamodeling* in Common Logic.

It seems that the modernized features of Common Logic are a reflection of the progress of modern computer languages. For example, the semantics of Common Logic introduced a new term, *universe of reference*, in addition to the universe of discourse in denotational semantics. A dialect is called *segregated* in which some names are *non-discourse names*, namely the denotations of the non-discourse names are in the universe of reference, but not in the universe of discourse in an interpretation. "Segregated dialects are commonly described to have a universe of discourse, without mentioning the universe of reference; and for non-segregated dialects the universes of discourse and of reference are identical. The distinction makes it possible to provide a single semantics which can cover both styles of dialect". The motivation of introducing the universe of reference and non-discourse names is likely to be for the provision against people who do not want to concern some terminologies out of concerning ontologies.[7] However, this notion is very akin to Smith's reference

[7] from private discussion on [55] at the conference.

and de-reference framework in Section 2.3. See Subsection 2.3.4. Therefore, we might be able to rephrase that the second language model of Common Lisp is non-segregated, and the third is segregated, in which symbols and internal structures are segregated. This language model will support to develop logic systems using objects in imperative computer languages which include symbols or variable names, structures, objects, whereas the meaning of “segregated” may be misunderstood from its introductory usage in Common Logic.

We claim that today’s OWL, including OWL 2, embraces some drawbacks for the practical usage. It seems to lead people into a blind alley without thinking what ontology is and how it should be represented as ontology description languages. Common Logic intends to be a common framework of concrete knowledge representation languages that are compatible with World Wide Web. Although actual dialect implementation of Common Logic is not emerging yet and no one can foresee the future of Common Logic, but we also hopefully desire that our experience for SWCLOS can contribute to establish Common Logic as one of practical knowledge representation and exchange language in the ISO standard.

Appendix A

Zermelo-Fraenkel Set Theory

The purpose of this appendix is to introduce Zermelo-Fraenkel Set Theory (ZF for short), which is considered the foundation of mathematical number theory today, to readers who are unfamiliar with it. The most part of the description is taken from a lecture of set theory at University of Amsterdam, "Zermelo-Fraenkel Set Theory" (H.C. Doets, 2002[15]) Chapter 1 and 2. However, readers must note that ZF is completed as a theory for mathematics, specifically for natural number, and does not presume any elements of ontology. The set theory in KIF ontology is introduced at Appendix B.

A Definitions and Axioms

Primitives. The axioms listed below use only two primitives: *set* and *membership*, namely $\in$.

Axiom 0. Note that we discuss only sets on ZF in this appendix. We start the discussion from the following two basic axioms.

(i) There exists at least one thing (in the universe).

(ii) Every thing is a set (in the universe).

Axiom 1 (Extensionality). Sets are completely determined by their elements, and it is expressed as:

$$\forall a \forall b[\forall x(x \in a \Leftrightarrow x \in b) \Rightarrow a = b] \tag{A.1}$$

This axiom means that for any a and for any b if every element of a is an element of b and vice versa, then a and b are equal. This axiom provides the definition of equality on sets.

Axiom 2 (Separation, Aussonderung). For an open well-formed formula φ,

$$\forall a \exists b \forall x[(x \in b) \Leftrightarrow (x \in a \wedge \varphi(x))] \quad \text{where } b \text{ does not occure in } \varphi(x). \tag{A.2}$$

Zermelo weakened Cantor's Comprehension Principle (6.1) to this Separation Axiom by postulating another set a for a set b in question. Note that, by Extensionality, the set b defined by the Separation Axiom is uniquely determined by postulated a and φ. It is written in usual notation as

$$b = \{x \in a \mid \varphi(x)\}. \tag{A.3}$$

If φ is a property of sets such that a set b exists, (unique set b by Extensionality), the elements of which are exactly the sets satisfying φ :

$$x \in b \Leftrightarrow \varphi(x), \tag{A.4}$$

then this set b is denoted by an expression $\{x \mid \varphi(x)\}$.

Classes. The concept of classes in ZF means a mere construct of the form $\{x \mid \varphi(x)\}$, where φ is a formula. In particular, sets are classes, but all classes are not sets.

Readers should note that this terminology *class* is unique in set theory and different from OWL and object oriented languages.

Proper Classes. There may be no set consisting exactly of the entities satisfying φ. In that case, the class $\{x \mid \varphi(x)\}$ is called *proper* in ZF.

Russell's class (the class of sets that do not belong to themselves) is an example of the proper class. The *universal class* is also a proper class in ZF but it is a set in Quine's New Foundations Set Theory (NF).

Universal Class. $\boldsymbol{R} = \{x \mid x = x\}$.

Russell's Class. $\mathcal{R} = \{x \mid x \notin x\}$.

By "a set $\{x \mid E(x)\}$" we mean there is a set a such that $\forall x[x \in a \Leftrightarrow E(x)]$.

Note that, by Axiom0 (ii) in ZF, proper classes simply don't exist, and so the use of abstractions $\{x \mid \varphi(x)\}$ must be regarded as a mere way of speech.

Empty Set. $\emptyset = \{x \in a \mid x \neq x\}$.
Note that, the empty set $\{x \mid x \neq x\}$ does not depend on the choice of a postulated set a.

Intersection. $a \cap b = \{x \in a \mid x \in b\} = \{x \mid x \in a \wedge x \in b\}$.

Pairing. $\forall a \forall b \exists c \forall x\, (x \in c \Leftrightarrow x = a \vee x = b)$.
Namely, paring defines a set c such that $\{x \mid x = a \vee x = b\}$. An *unordered pair* of a and b is postulated by this axiom, and it is denoted as $c = \{a, b\}$. Note that $\{a, b\} = \{b, a\}$ by this axiom.

Singleton. In case $a = b$, we obtain $\{a\} = \{a, a\}$.
The *singleton* is a special case of *unordered pair*.

Sumset. $\forall a \exists b \forall x[x \in b \Leftrightarrow \exists y(x \in y \wedge y \in a)]$.
Sumset of a is a set such that contains all elements of sets that are elements of a. That is a set such that $\{x \mid \exists y \in a(x \in y)\}$ and denoted as $\bigcup a$.

Union. $a \cup b = \bigcup\{a, b\} = \{x \mid x \in a \vee x \in b\}$ is called the *union* of a and b.
Note that $\{a_0, \ldots, a_n\} = \{a_0\} \cup \cdots \cup \{a_n\}$.

Subset. If $\forall y(y \in x \Rightarrow y \in a)$, x is called a subset of a and it is denoted by $x \subset a$.

Powerset. $\forall a \exists b \forall x(x \in b \Leftrightarrow x \subseteq a)$.
Here, b is called a powerset of a, and it is denoted in ZF by $b = \wp(a)$.

Infinity. $\exists a[\emptyset \in a \wedge (\forall x \in a(x \cup \{x\} \in a))]$.
There exists a set such that has initially the empty set as element, and composed of an element and the singleton of the element for every element. Obviously, this definition involves a recursive description, and it amounts to infinite number of elements. This axiom is also defined as another form for a set a, namely $\exists a[\emptyset \in a \wedge \forall x(x \in a \Rightarrow \exists y(y \in a \wedge \forall z(z \in y \Leftrightarrow z \in x \vee z = x)))]$.

Regularity, Foundation. $\forall a[a \neq \emptyset \Rightarrow \exists x(x \in a \wedge (x \cap a = \emptyset))]$.
This axiom means that any non-empty set has at least one element such that is disjoint with the set itself. This axiom ensures to forbid infinitely descending chains of membership. Note that $x \neq \{x\}$ in ZF.

B Remarks for ZF Set Theory from Ontology

The point of Zermelo-Fraenkel Set Theory is to avoid Russell's Paradox by weakening too powerful unrestrictive comprehension principle, which allows us to make too big sets such as Cantor's transfinite ordinal number, to Separation Axiom, which separates sets from classes in ZF such as *universal* or *Russell's* class. In the form of Separation Axiom, A in (6.3) or a in (A.2) may be a set or a class but Z in (6.3) or b in (A.2) that can be constructed by using Separation Axiom with an appropriate formula must be a set. In order to ensure the existence of such a set, Zermelo axiomatized the existence of only sets including the empty set with presuming to use the Infinite Axiom. From the viewpoint of ontology, Zermelo's sets are big enough, because it can handle sets that include a set of infinite number of members. However, it cannot handle any individuals in the universe of discourse. [62][60]

By contrast, from the viewpoint of mathematics, due to the unavailability of transfinite ordinal numbers ω, $\omega \cdot 2$, ..., etc., Zermelo's theory was regarded too weak. Fraenkel solved this problem by introducing Replacement Axiom with the Axiom of Choice. The Axiom of Choice was invented by Zermelo. Thus, the set theory for mathematical foundation by Zermelo and Fraenkel is called Zermelo-Fraenkel Set Theory (ZF) or Zermelo-Fraenkel Set Theory with Choice (ZFC), today. However, we do not need such extremely big sets for ontology.

Appendix B

Sets in KIF

The purpose of this appendix is to introduce KIF Set Theory, which is developed at KIF 3.0 [18] Chapter 7. In this theory, an ontologically basic set is composed of concrete objects rather than sets, whereas any set in ZF is composed of only sets including the empty set. Furthermore, a set in KIF is categorized into two distinct set categories, *bounded* set and *unbounded* set, and this distinction is critical to avoid Russell's Paradox.

A Rationale of Set Theory in KIF

Knowledge Interchange Format (KIF) is a language designed for use in the interchange of knowledge among disparate computer systems. The universe of discourse in KIF is defined as the set of all objects presumed or hypothesized to exist in the world in their own right. In Chapter 7 of KIF 3.0 [18], the description of KIF Sets starts as follows.

> "The formalization of sets of simple objects is a simple matter; but, when we begin to talk about sets of sets, the job becomes difficult due to the threat of paradoxes (like Russell's hypothesized set of all sets that do not contain themselves)."

It is obvious that KIF Set Theory has been developed in order to avoid the threat of Russell's Paradox. Then, it states that KIF adopted the set theory of von Neuman-Bernays-Gödel (NBG for short).

> "Fortunately, there is no shortage of mathematical theories for our use in KIF – various higher order logics, Zermelo-Fraenkel set theory, von Neuman-Bernays-Gödel set theory, Quine's variants on the previous two approaches, the more recently elaborated

> proposals by Feferman and Aczel, and so forth. In KIF, we have adopted a version of the von Neumann-Bernays-Gödel set theory."

The NBG is a conservative extension of 'Zermelo Fraenkel Set Thoery with the axiom of Choice' (ZFC). NBG distinguishes sets and classes as well as ZFC, and then the expressions of axioms are presented as a two-sorted theory for sets and classes, with lower case letters denoting variables ranging over sets, and upper case letters denoting variables ranging over classes. The four axioms in NBG, e.g., Paring, Union, Powerset, and Infinity, are identical to their ZFC counterparts. The Extensionality for sets is also identical to ZFC, and Extensionality for classes is similar to the Extensionality for sets and the Foundation is applied to classes.

KIF discusses how the set theory avoids the paradox. However, it should be noted here that the standard notion of *proper class* in set theories (ZF and NBG) is replaced by term *unbounded* set, and the standard notion of *class* is replaced by term *set* in KIF 3.0 documentation [18].

B Basic Concepts

"In KIF, a fundamental distinction is drawn between *individuals* and *sets*. A set is a collection of objects. An individual is any object that is not a set." [18] This statement is important to understand the difference between a set in mathematical set theories and a set in ontology. As shown at Axiom 0 in Appendix A, everything in the universe for a set theory is a set. By contrast, objects in KIF are individuals, sets of individuals, and sets of sets.

Sets and Individuals. The sentence $individual(x)$ is true if and only if the object denoted by x is an individual. The sentence $set(x)$ is true if and only if the object denoted by x is a set. Individuals and sets are exhaustive and mutually disjoint. That is;

$$\forall x(set(x) \Leftrightarrow \neg individual(x)) \tag{B.1}$$

$$\forall x(individual(x) \Leftrightarrow \neg set(x)) \tag{B.2}$$

$$\forall x \forall s(\{x \mid individual(x)\} \cup \{s \mid set(s)\}) = \boldsymbol{R} \tag{B.3}$$

$$\forall x \forall s(\{x \mid individual(x)\} \cap \{s \mid set(s)\}) = \emptyset. \tag{B.4}$$

In addition to distinguishing *individuals* and *sets*, the category of *bounded* and *unbounded* objects are introduced.

> "A distinction is also drawn between objects that are *bounded* and those that are *unbounded*. This distinction is orthogonal to the distinction between individuals and sets. There are bounded individuals and unbounded individuals. There are bounded sets and unbounded sets."

Bounded and Unbounded. The sentence $bounded(x)$ is true if and only if the object denoted by x is bounded. The sentence $unbounded(x)$ is true if and only if the object denoted by x is unbounded. Boundedness and unboundedness are exhaustive and mutually disjoint.

$$\forall x(bounded(x) \Leftrightarrow \neg unbounded(x)) \tag{B.5}$$

$$\forall x(unbounded(x) \Leftrightarrow \neg bounded(x)) \tag{B.6}$$

$$\forall x_1 \forall x_2[\{x_1 \mid bounded(x_1)\} \cup \{x_2 \mid unbounded(x_2)\}] = \boldsymbol{R} \tag{B.7}$$

$$\forall x_1 \forall x_2[\{x_1 \mid bounded(x_1)\} \cap \{x_2 \mid unbounded(x_2)\}] = \emptyset \tag{B.8}$$

A set can have members, but an individual cannot. A bounded objects can be a member of sets, but an unbounded object cannot. The finite sets are bounded as shown below. So, a finite set and a bounded individual can be a member of sets. Thus, an interesting thing in the ontological view is what individual is bounded and what individual is unbounded. However, the KIF documentation did not state anything about it. Instead it just provides a unary predicate `bounded` and `unbounded` as well as `set` and `individual`.

As mentioned above, an object can be a member of another object if and only if the former is bounded and the latter is a set.

Axiom 1 (Membership).

$$\forall x \exists s[x \in s \Leftrightarrow bounded(x) \wedge set(s)] \tag{B.9}$$

C Sets

Many axioms in ZF and NBG are also axiomatized for sets in KIF. In addition, the *axiom of choice* is modified for KIF. The Axiom of Choice in ZF is described in the following statement (Chapter 5 in [15]).

> *Every set has a choice function.*

Supposing in ZF let A be a set (of sets), a choice function f for A is a mapping from domain $A-\emptyset$ (for a non-empty element) to a set such that every element satisfies $f(x) \in x$ (choose one representative

from each elemental set of A). The choice function in KIF chooses every element that is uniquely associated to each bounded element, and it asserts that there is a set that associates every bounded set with a distinguished element of that set.

Axiom 2 (Choice).

$$\begin{aligned}\exists s[&\forall \tau \exists x \exists y(\tau \in s \Rightarrow \{x, y\}) \\ &\wedge \ \forall x \forall y \forall z((\{x, y\} \in s \wedge \{x, z\} \in s) \Rightarrow y = z) \\ &\wedge \ \forall u(\mathit{bounded}(u) \wedge u \neq \emptyset \Rightarrow \exists v(v \in u \wedge \{u, v\} \in s))]\end{aligned} \tag{B.10}$$

D Boundedness

The key difference between bounded and unbounded objects is that the former can be members of other sets while the latter cannot. This fact establishes a necessary and sufficient test for boundedness – an object is bounded just in case it is a member of a set. However, this is not very helpful to determine whether or not an object is bounded based on other properties. The following axioms and description is useful to decide the boundedness.

Axiom 3 (Bounded Finite Set). Any finite set of bounded sets is itself a bounded set.

$$\forall s_0 \ldots \forall s_k(\bigwedge_{i=0,\ldots,k} \mathit{bounded}(s_i) \Rightarrow \mathit{bounded}(\{s_0, \ldots, s_k\}) \tag{B.11}$$

Axiom 4 (Bounded Subset). The set of all subsets of a bounded set is also a bounded set.

$$\forall v[\mathit{bounded}(v) \Rightarrow \forall u(\mathit{bounded}(\{u \mid u \subset v\})] \tag{B.12}$$

Axiom 5 (Bounded Union). The sumset of any bounded set of bounded sets is also a bounded set.

$$\forall u[\mathit{bounded}(u) \wedge \forall x(x \in u \Rightarrow \mathit{bounded}(x)) \Rightarrow \mathit{bounded}(\bigcup u)] \tag{B.13}$$

Here, the notation $\bigcup$ represents a sumset, see Appendix A. Since every finite set is bounded, this allows us to conclude that the union of any finite number of bounded sets is a bounded set.

Axiom 6 (Bounded Intersection). The intersection of a bounded set and any other set is a bounded set.

$$\forall u \forall s \forall x[\mathit{bounded}(u) \wedge x \in s \Rightarrow \mathit{bounded}(\{x \mid x \in u \wedge x \in s\})] \tag{B.14}$$

So long as one of the sets defining the intersection is bounded, the resulting set is bounded.

Axiom 7 (Bounded Infinity). There is a bounded set containing a set, then a set that properly contains that set, and then a third set that properly contains the second set, and so forth. In short, there is at least one bounded set of infinite cardinality.

$$\exists u[\mathit{bounded}(u) \wedge u \neq \emptyset \wedge \forall x(x \in u \Rightarrow \exists y(x \subset y \wedge y \in u))] \tag{B.15}$$

E Paradoxes

The paradoxes appear only when we try to define set primitives that are too big and too powerful. We might consider defining the term $\{x \mid \varphi(x)\}$ to mean simply the set of all objects denoted by x for any assignment of the free variables of x that satisfies $\varphi(x)$. Unfortunately, these two definitions quickly lead to paradoxes, as mentioned earlier in Section 6.1.

KIF describes the harmful influence of paradoxes as:

> "One of the goals in the design of KIF is that it has a clearly specified model-theoretic semantics in terms of which the concepts of entailment, equivalence, consistency, soundness and completeness can be defined. If the paradoxes are allowed to persist in principle, even if they are easy to avoid in practice, the consequence would be that no KIF theory would be true in any model. Definitions couched in terms of models would be trivialized, becoming useless. All sentences would be entailed by any theory, any two theories would be equivalent, no theory would be consistent, every possible inference rule would be sound, and so on."

In the NGB version of KIF, these paradoxes are avoided by replacing the principle of unrestricted set abstraction with the *principle of restricted set abstraction* by bounded objects as follows.

$$x \in \{\nu \mid \varphi(\nu)\} \Leftrightarrow \mathit{bounded}(x) \wedge \varphi_{\nu/x} \tag{B.16}$$

where $\varphi_{\nu/x}$ stands for the result of substitution of term x for all free occurrences of ν in $\varphi(\nu)$.

Thus, eventually KIF can avoid Russell's Paradox by introducing boundedness and combining it with any formula. The KIF document concludes as follows in Chapter 7.

> "With this principle, there are two reasons why something may be excluded from a set $\{\nu \mid \varphi(\nu)\}$. It may fail to be a member because it does not satisfy the defining

condition $\varphi(v)$, or it may be excluded because it is an unbounded object. Conditioning the membership of objects in this set on their boundedness effectively eliminates the paradoxes."

Appendix C

Ramified Type Theory

The purpose of this appendix is to introduce Russell's Ramified Type Theory (RTT), which is the first genuine type theory developed in 'Principia Mathematica' (PM) for the purpose of avoiding Russell's paradox in case of the axiomatization of mathematical system. In this appendix, we specialize the description of Laan [44] and Kamareddine et al. [28] for RTT into unary predicate (for class extension) and binary predicate (for property extension) of higher order logic along with the definition of RDF Semantic Theory.

A Vicious Circle Principle

Bertrand Russell and Alfred N. Whitehead conceived that the so-called Russell's Paradox is one of many paradoxes all of which are caused by vicious circle.

> An analysis of the paradoxes to be avoided shows that they all result from a certain kind of vicious circle. (Principia Mathematica [77], pp.39)

Therefore, they settled *vicious circle principle* in order to avoid the paradoxes.

> [...] "Whatever involves *all* of a collection must not be one of the collection"; or, conversely: "If, provided a certain collection had a total, it would have members only definable in terms of that total, then the said collection has no total." We shall call this the "vicious-circle principle," [...] (PM [77], pp.39)

The universe of discourse is exactly 'a total' and 'no total' mentioned in PM, and the universal class or rdfs:Resource is a class that denotes the totality. Russell invented Ramified Type Theory

in order to avoid the Russell's Paradox and take the totality into account. However, obscurities in their formalization remained in the description from the today's viewpoint that is mathematically rigorous, and then researchers re-formulated and re-interpreted more precisely the Ramified Type Theory later[1].

In the following sections, according to the description by Kamareddine et al. [28], we simplify the Ramified Type Theory only into unary and binary predicate, adapting the theory to RDF Semantic Theory.

B Propositional Function

In PM, the concept of *propositional function* was introduced onto the logic by Frege instead of set theories.

> "By a 'propositional function' we mean something which contains a variable x, and expresses a *proposition* as soon as a value is assigned to x." (PM, pp.41)

For example, "Obama is Human" or "$Human(Obama)$" is a proposition. "$Human(x)$" and "$x(Obama)$" is a propositional function, respectively.

The following metavariables for individuals, variables, and functions are distinctive as symbol. However, actual vocabulary of individuals, variables, and functions are not distinctive, and function arguments, i.e., i, j and k, are metavariables that run over individuals, variables, and functions.

- $\mathcal{A}$ is a vocabulary of *individuals*. $a, a_1, a_2, \ldots$ and $b, b_1, b_2, \ldots$ are symbols for vocabulary $\mathcal{A}$. $a_i^{\mathcal{I}}$ and $b_i^{\mathcal{I}}$ is an expression for the denotation of a_i and b_i, respectively.
- $\mathcal{V}$ is a vocabulary of *variables*. $x, x_1, x_2, \ldots$ and $y, y_1, y_2, \ldots$ are symbols for vocabulary $\mathcal{V}$. $x_i^{\mathcal{I}}$ and $y_i^{\mathcal{I}}$ is an expression for the denotation of x_i and y_i, respectively.
- C is a vocabulary of *unary predicates*. $\mathcal{R}$ is a vocabulary of *binary predicates*. $C, C_1, C_2, \ldots$ are symbols for vocabulary C. $R, R_1, R_2, \ldots$ are symbols for vocabulary $\mathcal{R}$. $C_i^{\mathcal{I}}$ is an expression for the denotation of C_i. $R_i^{\mathcal{I}}$ is an expression for the denotation of R_i.

(Atomic Proposition) $C(a)$ is called (unary) atomic proposition. $R(a, b)$ is called (binary) atomic proposition. $\mathcal{I}(C(a)) = C^{\mathcal{I}}(a^{\mathcal{I}})$ denotes an interpretation $\mathcal{I}$ of $C(a)$. $\mathcal{I}(R(a, b)) = R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}})$ denotes an interpretation $\mathcal{I}$ of $R(a, b)$.

[1] See Kamareddine et al. [28], pp.21.

Propositional functions in PM are generated from atomic propositions by two means, namely, i) the use of logical connectives and quantifiers, and ii) abstraction from earlier generated propositional functions, using the abstraction principle.
(Propositional function) We define a collection of propositional functions, $\mathcal{P}$. Such as:

1. If $i, j \in \mathcal{A} \cup \mathcal{V}$, then $C(i), R(i, j) \in \mathcal{P}$.
2. If $f, g \in \mathcal{P}$, then $f \vee g \in \mathcal{P}$ and $\neg f \in \mathcal{P}$.
3. If $f \in \mathcal{P}$ and x is free in f, then $\forall x[f] \in \mathcal{P}$.
4. If $k_1, k_2 \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$, then $C(k_1), R(k_1, k_2) \in \mathcal{P}$.
5. All propositional functions can be constructed by using the construction rules 1, 2, 3 and 4 above.

Note that rule 2 and 3 above allow us to describe not only disjunction ($\vee$) but also conjunction ($\wedge$) due to the negation ($\neg$)[2]. In case of rule 1 through 3, we stay in first order logic, rule 4 allows us to expand propositional functions to higher order.
(Proposition) A propositional function f is a *proposition* if there is no free variable in f.

So far, we do not set *vicious circle principle* yet. For example, we can define the so-called Russell's set $\neg z(z)$[3] as propositional function.

C Ramified Type

Here, in order to avoid the paradox, ramified types are introduced. while there is no definitions of "type" in PM, Kamareddine et al. pointed out that the definition of "the same type" in $*9{\cdot}131$ in PM.

> $*9{\cdot}131$ *Definition of "being of the same type."* The following is a step-by-step definition, the definition for higher types presupposing that for lower types. We say that u and v "are of the same type" if (1) both are individuals, (2) both are elementary functions taking arguments of the same type, (3) u is a function and v is its negation, (4) u is $\phi\hat{x}$[4] or $\psi\hat{x}$, and v is $\psi\hat{x} \vee \psi\hat{x}$, where $\phi\hat{x}$ and $\psi\hat{x}$ are elementary functions, (5) u is

[2] We assume the law of excluded middle.
[3] It corresponds to $x \notin x$, which causes Russell's Paradox.
[4] The notation $\phi\hat{x}$ in PM denotes a propositional function that has x as a free variable. Similary, $\phi(\hat{x}, \hat{y})$ is used for the representation of a propositional function ϕ that has two free variables x and y amongst its free variables.

> $\forall y[\phi(\hat{x}, y)]$ and v is $\forall z[\psi(\hat{x}, z)]$, where $\phi(\hat{x}, \hat{y})$ and $\psi(\hat{x}, \hat{y})$ are of the same type, (6) both are elementary propositions, (7) u is a proposition and v is $\neg u$, or (8) u is $\forall x[\phi x]$ and v is $\forall y[\psi y]$, where $\phi\hat{x}$ and $\psi\hat{x}$ are of the same type. (PM [77],Vol.1,pp.138)

Here, an elementary function is a propositional function that takes only elemental propositions as its value. An elemental proposition is a proposition that does not involve any variables. See [77],pp.95.

According to these rules for "the same type", Kamareddine et al. pointed out that $z() \vee \neg z()$ is not of the same type as $z()$.

(Simple type) firstly simple type is defined such as:

1. 0 is a simple type;
2. If $t_1, \ldots, t_n$ are simple types, then also $(t_1, \ldots, t_n)$ is a simple type. $n = 0$ is allowed: then we obtain the simple type ();
3. All imple types can be constructed using above rules 1 and 2.

The type () stands for the type of the proposition, and the type 0 stands for the type of the individuals. Then, the unary propositional function that takes one individual as argument should have type (0).

It is also pointed that $C(a)$ and $\forall z : ()[z() \vee \neg z()]$ are of a different level, because the former is an atomic proposition, while the latter is based on the propositional function $z() \vee \neg z()$. Note that the expression of propositional function $z() \vee \neg z()$ involves an arbitrary propositions z, therefore $\forall z : ()[z() \vee \neg z()]$ denotes all interpretations quantified over the universe of discourse. However, according to the vicious circle principle, $\forall z : ()[z() \vee \neg z()]$ itself cannot belong to this collection of propositions. This problem is solved by dividing types into *orders*. We start *ramified types* from type 0 with order 0 for individuals.

1. 0^0 is a ramified type;
2. If $t_1^{m_1}, \ldots, t_n^{m_n}$ are ramified types, then $(t_1^{m_1}, \ldots, t_n^{m_n})^N$ is a ramified type, where $N > max(m_1, \ldots, m_n)$;
3. All ramified types can be constructed using rules 1 and 2.

If t^m is a ramified type, then m is called the *order* of t^m.

Amongst the ramified types, ones of $N = max(m_1, \ldots, m_n)+1$ is called *predicative*. For example, for $Obama \in \mathcal{A}$, individual $Obama^{\mathcal{I}}$ has type 0^0, then $Human^{\mathcal{I}}(Obama^{\mathcal{I}})$ has type $(0^0)^1$. Therefore, $rdfs{:}Class^{\mathcal{I}}(Human^{\mathcal{I}})$ has type $((0^0)^1)^2$.

We conceive that RDF classes in RDF Semantics are equivalent to the unary predicates and they have a predicative type. Thus, $x^{\mathcal{I}} \in CEXT^{\mathcal{I}}(C^{\mathcal{I}})$ is equivalent to $C^{\mathcal{I}}(x^{\mathcal{I}})$, and if the order of $x^{\mathcal{I}}$ is n, then the order of $C^{\mathcal{I}}$ is $n+1$. Thus, for the equation on the universal metaclass (2.12) or (6.13), if the order of $rdfs:Class^{\mathcal{I}}$ at the left hand side is n, then the order of $rdfs:Class^{\mathcal{I}}$ at the right hand side is $n+1$. Both are the same in appearance, but they must be distinguished by the orders in Ramified Type Theory. Therefore, the descendant of membership on rdfs:Class is terminated from an arbitrary n to the bottom order 2 of rdfs:Class such as $n \to n-1 \to n-2 \to \cdots \to 2$, because RDF properties, e.g., $rdfs:comment^{\mathcal{I}}$ and $rdfs:label^{\mathcal{I}}$, have type 0^0, and then $rdf:Property^{\mathcal{I}}$ has type $(0^0)^1$, then $rdfs:Class^{\mathcal{I}}(rdf:Property^{\mathcal{I}})$ has type $((0^0)^1)^2$.

[This page intentionally left blank]

Appendix D

List of Published Papers

【ジャーナル】

小出 誠二，武田 英明：OWL 意味論に基づく CLOS オブジェクト指向プログラミング，コンピュータソフトウェア，2011-5 月予定，岩波，25 ページ

【解説論文】

小出誠二, 武田英明：人工知能用言語 Lisp の今と将来，人工知能学会誌， 24(5)，681-690，2009，10 ページ

【国際会議（査読つき）】

Seiji Koide and Hideaki Takeda, Common Languages for Web Semantics, Evaluation of Novel Approaches to Software Engineering (CCIS), Lecture Notes, Springer-Verlag (2011), 16 pages.

Seiji Koide and Hideaki Takeda, Common Languages for Semantic WWW – Beyond RDF and OWL –, 5th Int. Conf. Evaluation of Novel Approaches to Software Eng. (ENASE2010), 10 pages.

Seiji Koide and Hideaki Takeda, Meta-circularity and MOP in Common Lisp for OWL Full, ELW '09: Proceedings of the 6th European Lisp Workshop, 28–34, ACM. 7 pages.

Seiji Koide and Hideaki Takeda, Formulation of Hierarchical Task Network Service (De)composition, First International Joint Workshop SMR2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web, Workshop at ISWC2007+ASWC2007, 107–121, 15 pages

Seiji Koide and Hideaki Takeda, OWL-Full Reasoning from an Object Oriented Perspective, Asian Semantic Web Conf. (ASWC200), 263–277, Springer. 15 pages

Seiji Koide and Hideaki Takeda, MetaModeling in OOP, MOF, RDFS, and OWL, 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006) at the 5th International Semantic Web Conference (ISWC 2006), 14 pages

【チュートリアルその他】

Seiji Koide, SWCLOS: A Semantic Web Processor on CLOS, Europian Lisp Workshop, Tutorial, 2009

Seiji Koide, SWCLOS User's Manual, National Institute of Informatics Technical Report (NII-2009-014E), 2009, 92 pages

Seiji Koide: Semantic Web Services and OOP – toward Unified Services –, SWO, 2006

小出 誠二，セマンティックウェブサービスによるロケット打上支援システム，日本人工知能学会第２０回 AI シンポジウム，2005

Seiji Koide, SWCLOS: Semantic Web Processing in CLOS, International Lisp Conference, Tutorial, 2005

Bibliography

[1] Ontology Definition Metamodel. Standard document, OMG, May 2009. OMG Document Number: formal/2009-05-01.

[2] José Júlio Alferes, Matthias Knorr, and Terrance Swift. Queries to hybrid mknf knowledge bases through oracular tabling. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 1–16, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damásio, and Gerd Wagner. Stable model theory for extended RDF ontologies. In *ISWC 2005*, volume 3729 of *LNCS*, pages 21–36, 2005.

[4] Colin Atkinson. Unifying MDA and knowledge representation technologies. In *1st International Workshop on the Model-Driven Semantic Web (MDSW2004) Enabling Knowledge Representation and MDA Technologies to Work Together*, 2004.

[5] Franz Baarder and Werner Nutt. *The Description Logic Handbook*, chapter 2. Basic Description Logics, pages 43–95. Cambridge, 2003.

[6] Alex Borgida and Ronald J. Brachman. *The Description Logic Handbook*, chapter 10. Conceptual Modeling with Description Logics, pages 349–372. Cambridge, 2003.

[7] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004.

[8] Saartje Brockmans, Robert M. Colomb, Peter Haasea, Elisa F. Kendall, Evan K. Wallace, Chris Welty, and Guo Tong Xie. A model driven approach for building OWL DL and OWL Full ontologies. In *ISWC 2006*, volume 4273 of *LNCS*, pages 187–200, 2006.

[9] Jeremy Carroll, Ivan Herman, and Peter F. Patel-Schneider. Owl 2 web ontology language rdf-based semantics. W3C Candidate Recommendation, June 2009. http://www.w3.org/TR/owl2-rdf-based-semantics/.

[10] Jeremy J. Carroll and Dave Turner. The consistency of OWL Full (with proofs). Technical Report HPL-2008-59, HP Laboratories, 2008.

[11] Robert M. Colomb, Anna Gerber, and Michael Lawley. Issues in mapping metamodels in the ontology development metamodel. In *Proc. Model-Driven Semantic Web*, 9 2004.

[12] Jos de Bruijn, Enrico Franconi, and Sergio Tessaris. Logical reconstruction of RDF and ontology languages. In *Principles and Practice of Semantic Web Reasoning*, volume 3703 of *LNCS*, pages 65–71. Springer, 2005.

[13] Jos de Bruijn and Stijn Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In K. Aberer at al., editor, *ISWC/ASWC 2007, The Semantic Web*, volume 4825 of *LNCS*, pages 86–99. Springer, 2007.

[14] Dragan Djurić, Dragan Gašević, and Vladan Devedžić. Ontology modeling and MDA. *Journal of Object Technology*, 4(1):109–128, January-February 2005.

[15] H.C. Doets. Zermelo-Fraenkel Set Theory. Lecture Note, April 2002. `http://staff.science.uva.nl/~vervoort/AST/ast.pdf`.

[16] douglas foxvog. Instances of instances modeled via higher-order classes. In *Foundational Aspects of Ontologies (FOnt 2005)*, pages 46–54, Koblenz, Germany, September 2005. 28th German Conference on Artificial Intelligence. ISSN 1860-4471.

[17] Gilles Dowek. *Higher-Order Unification and Matching, HANDBOOK OF AUTOMATED REASONING*, chapter 16, pages 1009–1062. Elsevier, 2001.

[18] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format version 3.0 reference manual, 1994.

[19] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semantics*, 6:309–322, 2008.

[20] Nicola Guarino. Some ontological principles for designing upper level lexical resources. In A. Rubio, N. Gallardo, R. Castro, and A. Tejada, editors, *the First International Conference*

on Lexical Resources and Evaluation, pages 527–534, Granada, Spain, May 1998. ELRA - European Language Resources Association.

[21] R. V. Guha and Patric Hayes. Lbase: Semantics for languages of the semantic web. Note, W3C, October 2003.

[22] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[23] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with racer + nRQL. In *Proceedings of the Workshop on Description Logics 2004 (ADL2004)*, 2004.

[24] Harry Halpin, Patrick J. Hayes, James P. McCusker, Deborah L. Mcguinness, and Henry S. Thompson. When owl:sameas ins't the same: An analysis of identity in linked data. In *ISWC 2010*, 2010.

[25] Patrick Hayes and Brian McBride. RDF Semantics. W3C Recommendation, Feb. 2004. http://www.w3.org/TR/rdf-mt/.

[26] Patrick Hayes and Christopher Menzel. A semantics for the knowledge interchange format. In *In IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*, 2001.

[27] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. Owl 2 web ontology language primer. W3C Recommendation, October 2009.

[28] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. *A Modern Perspective on Type Theory*. Kluwer Academic Publishers, 2004.

[29] Ken Kaneiwa and Riichiro Mizoguchi. An order-sorted quantified modal logic for meta-ontology. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005)*, volume LNCS 3702, pages 169–184. Springer Verlag, 2005.

[30] Heiko Kattenstroth. *Knowledge-Management with OWL and F-logic: A Combination of Desciption Logic Reasoning with F-Logic Rules*. PhD thesis, Univ. Goettingen, 2007.

[31] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[32] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[33] Michael Kifer and James Wu. A logic for object-oriented logic programming. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '89)*, 1989.

[34] Graham Klyne, Jeremy J. Carroll, and Brian McBride. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation 10 February 2004, February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts/.

[35] Holger Knublauch, Daniel Oberle, Phil Tetlow, and Evan Wallace. A semantic web primer for object-oriented software developers. W3C Working Group Note, http://www.w3.org/TR/sw-oosd-primer/, March 2006.

[36] Paul Kogut, Stephen Cranefield, Lewis Hart, Mark Dutra, Kenneth Baclawski, Mieczyslaw Kokar, and Jeffrey Smith. UML for ontology development. *The Knowledge Engineering Review*, 17(1):61–64, 2002.

[37] Seiji Koide. Swclos user's manual. Technical Report NII-2009-014E, National Institute of Informatics, Oct. 2009. communicated by Hideaki Takeda.

[38] Seiji Koide, Jans Aasman, and Steve Haflich. OWL vs. object oriented programming. In *Workshop on Semantic Web Enabled Software Engineering (SWESE) at the 4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, November 2005.

[39] Seiji Koide and Masanori Kawamura. SWCLOS: A semantic web processor on common lisp object system. In *3rd International Semantic Web Conference (ISWC2004), Demos*, 2004.

[40] Seiji Koide and Hideaki Takeda. Metamodeling in OOP, MOF, RDFS, and OWL. In *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006) at the 5th International Semantic Web Conference (ISWC 2006)*, Athens, GA, U.S.A., November 2006.

[41] Seiji Koide and Hideaki Takeda. OWL-Full reasoning from an object oriented perspective. In *Asian Semantic Web Conf., ASWC2006*, pages 263–277. Springer, 2006.

[42] Seiji Koide and Hideaki Takeda. Meta-circularity and mop in common lisp for owl full. In *ELW '09: Proceedings of the 6th European Lisp Workshop*, pages 28–34, New York, NY, USA, 2009. ACM.

[43] Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura, and Riichiro Mizoguchi. Role representation model using owl and swrl. In *Proc. of 2nd Workshop on Roles and Relationships in Object Oriented Programming, Multiagent Systems, and Ontologies*, Berlin, July 2007.

[44] Twan Laan. A formalization of the ramified type theory. Technical report, 1995.

[45] Li Ma, Yang Yang, Zhaoming Qiu, Guotong Xie, Yue Pan, and Shengping Liu. Towards a complete owl ontology benchmark. In Y. Sure and J. Domingue, editors, *Proc. ESWC 2006*, number 4011 in LNCS, pages 125–139. Springer, 2006.

[46] Pattie Maes. Concepts and experiments in computational reflection. In *OOPSLA '87 Proceedings*, pages 147–155. ACM, October 1987.

[47] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, and Naveen Srinivasan Katia Sycara. OWL-S: Semantic markup for web services. http://www.w3.org/Submission/2004/ SUBM-OWL-S-20041122/, 2004. W3C.

[48] Drew McDermott. Tarskian semantics, or no notation without denotation! *Cognitive Science*, 2:277–282, July-September 1978.

[49] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004, February 2004. http://www.w3.org/TR/owl-features/.

[50] Georgios Meditskos and Nick Bassiliades. Towards an object-oriented reasoning system for owl. In *Proc. OWL: Experiences and Directions Workshop*, pages 11–12, 2005.

[51] Shohei Misono, Seiji Koide, Norikazu Shimada, Masanori Kawamura, and Susumu Nagano. Distributed collaborative decision support system for rocket launch operation. In *IEEE/ASME International Conference on Advanced Intelligent Mechtronics (AIM2005), WB3-01*, pages 1318–1323, 2005.

[52] Riichiro Mizoguchi. Tutorial on ontological engineering - part 2: Ontology development, tools and languages. *New Generation Computing*, 22(1):61–96, 2004.

[53] Riichiro Mizoguchi, Eiichi Sunagawa, Kouji Kozaki, and Yoshinobu Kitamura. The model of roles within an ontology development tool: Hozo. *Appl. Ontol.*, 2(2):159–179, 2007.

[54] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241, 2006.

[55] Fabian Neuhaus. The semantics of modules in common logic. In Barry Smith, Riichiro Mizoguchi, and Sumio Nakagawa, editors, *Interdisciplinary Ontology*, volume 3, pages 107–117. Open Research Centre for Logic and Formal Ontology, Keio University, February 2010.

[56] Andreas Paepcke, editor. *Object-Oriented Programming - The CLOS Perspective*. MIT Press, 1993.

[57] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, February 2004.

[58] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax section 5. rdf-compatible model-theoretic semantics. W3C Recommendation, http://www.w3.org/TR/owl-semantics/rdfs.html, February 2004.

[59] Peter F. Patel-Schneider and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax section 3. Direct Model-Theoretic Semantics. W3C Recomendation, 2 2004. http://www.w3.org/TR/owl-semantics/direct.html.

[60] Michael Potter. *Set Theory and its Philosophy*. Oxford, 2004.

[61] Colin Puleston, Bijan Parsia, James Cunningham, and Alan Rector. Integrating object-oriented and ontological representations: A case study in java and OWL. In *Proc. ISWC 2008*, volume 5318 of *LNCS*, pages 130–145. Springer, 2008.

[62] Willard van Orman Quine. *Set Theory and its Logic*. Harvard, 1963.

[63] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpura, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics*, 5:51–53, June 2007.

[64] Brian Cantwell Smith. Reflection and semantics in Lisp. In *POPL '84: Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 23–35, New York, NY, USA, 1984. ACM Press.

[65] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-guide/.

[66] John F. Sowa. Top-level ontological categories. *Int. J. Hum.-Comput. Stud.*, 43(5-6):669–685, 1995.

[67] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA., August 1999.

[68] Guy L. Steele, Jr. *Common Lisp The Language*. Digital Press, 1984.

[69] Guy L. Steele, Jr. *Common Lisp The Language 2nd edition*. Digital Press, 1990.

[70] Hideaki Takeda, K. Iino, and Toyoaki Nishida. Agent organization and communication with multiple ontologies. *Int. J. Cooperative Inf. Syst.*, 4(4):321–338, 1995.

[71] Alfred Tarski. *Introduction to Logic*. Dover, 1946/1995. This book is an extended edition of the book of title "On Mathematical logic and Deductive Method," which appeared at 1936 in Polish.

[72] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. *Web Semantics*, 3:79–115, 2005.

[73] Phil Tetlow, Jeff Z. Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. W3C Editors' Draft, Feb 2006. http://www.w3.org/2001/sw/BestPractices/SE/ODA/.

[74] David Turner and Jeremy J. Carroll. Comparing owl semantics. Technical Report HPL-2007-146, Digital Media Systems Laboratory, HP Laboratories Bristol, 2007.

[75] Sui-Yu Wang, Yuanbo Guo, Abir Qasem, and Jeff Heflin. Rapid benchmarking for semantic web knowledge base system. In Y. Gil et al., editor, *Proc. ISWC 2005*, volume 3729 of *LNCS*, pages 758–772. Springer, 2005.

[76] Timo Weithöner, Thorsten Liebig, Marko Luther, Sebastian Böhm, Friedrich Henke, and Olaf Noppens. Real-world reasoning with owl. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, ESWC '07, pages 296–310, Berlin, Heidelberg, 2007. Springer-Verlag.

[77] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume 1. Merchant Books, 1910.

Index