Adrian Groza

ONTOLOGY ENGINEERING WITH RACERPRO

An activity-based approach

U.T. PRESS
Cluj-Napoca, 2014
ISBN 978-973-662-991-4

Editura U.T.PRESS
Str.Observatorului nr. 34
C.P.42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999 / Fax: 0264 - 430.408
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Director: Prof.dr.ing. Daniela Manea
Consilier editorial: Ing. Călin D. Câmpean

Recenzia: Prof.dr. ing. Ioan Alfred Leţia
Prof. dr. ing. Alin Suciu

ISBN 978-973-662-991-4
Bun de tipar: 31.07.2014
Tiraj: 100 exemplare

Ontology Engineering with RacerPro

An Activity-Based Approach

Adrian Groza
Intelligent Systems Group
Adrian.Groza@cs.utcluj.ro

Department of Computer Science
Technical University of Cluj-Napoca

Contents

Prologue

Active learning is described by Silberman [49]*"Above all, students need to 'do it' - figure things out by themselves, come up with examples, try out skills, and do assignments that depend on the knowledge they already have or must aquire"*. Among the various implementations of active learning in computer science (i.e. [34, 54, 19]), this manual is an implementation of the active-learning approach by offering a large collection of exercises in the domain of ontology engineering. In line with a lab-based teaching strategy [26], the exercises are designed to enable students to explore their problem learning strategies, to express various solutions to the same problem, or to explore new topics in Semantic Web. Consequently, the theoretical part is reduced, while focus is on solving various issues related to description logics, ontology engineering or ontology evaluation.

This manual is based on the RacerPro knowledge representation and reasoning system [24]. In our view, RacerPro and the corresponding KRSS syntax for Description Logic axioms are powerful technical instrumentation that support students to exercise ontology engineering behind the basic capabilities provided by GUI-based ontology editors. Specifically, the KRSS syntax facilitates the understanding of the axioms developed by learners, while RacerPro provides a wide set of alternative primitives to introduce concepts, roles, constraints on these roles, debugging axioms and query the knowledge base.

The manual is designed for the Knowledge-Based Systems course taught at Computer Science Department, Technical University of Cluj-Napoca, Romania. However, it can serve all learners of description logic, ontology engineering, or, from a larger perspective, of Semantic Web. We argue that, due the logic-oriented nature of description logic and due to the simplicity of the KRSS syntax, students with mathematical and logical background should not encounter any difficulties when using this manual.

The manual consists of two parts. The first part aims to guide the students throughout various issues in ontology engineering by means of examples, explanations, tutorials, and exercises. The second part describes two ontologies developed in the KRSS syntax. In what follows, I briefly describe the content of each chapter of this manual.

Chapter 1 - Introducing Ontologies in the KRSS Syntax. This chapter introduces the technical instrumentation used throughout the paper. The KRSS syntax for developing ontologies is introduced by means of two basic ontologies: family and pizza. The RacerPro server and the RacerPorter client are described as the main technology used to support ontology engineer throughout this manual.

Chapter 2 - Reusing Ontologies. The chapter aims to familiarise the learner with the current development of Semantic Web. The student is encouraged to explore upper level ontologies and domain ontologies by means of semantic search engines and various ontology repositories. The selected ontologies are converted in the KRSS syntax and analysed for possible reuse. Competency questions are introduced to narrow the scope of the ontology to be developed.

Chapter 3 - Defining Concepts. This chapter starts by introducing the primitives used to formalised concepts in an ontology. The learners are familiarised with the subsumption relation between concepts. Then, the chapter presents queries available to retrieve information about the concepts in a knowledge base. The last part exemplifies how to handle terminological boxes, as the main building blocks of grouping axioms in a large ontology.

Chapter 4 - Defining Roles. This chapter introduces role axioms, role properties, and role queries. The role axioms specify the relationships between concepts. The role properties constraint these relationships by specifying some mathematical properties: a role can be transitive, symmetric, asymmetric, reflexive, irreflexive, functional or it can

have a specified domain and range. These mathematic constraints have an important role when reasoning on an ontology. The last conceptual instrumentation introduced in this chapter regards queries used to retrieve information about the existing roles in an ontology.

Chapter 5 - Populating Ontologies. This chapter presents different ways to introduce assertions: concept assertions, role assertions, attribute assertions and constraints. It also discusses various assumptions when reasoning on ontologies: the open and closed words assumptions and the unique name assumption.

Chapter 6 - Rules on Top of Ontology. This chapter introduces the technical instrumentation provided by RacerPro when dealing with rules. Rules are required to augment the expressivity limitation of description logic. After analysing the rules lifecycle, the event recognition capability of RacerPro is also presented.

Chapter 7 - Ontology Design Patterns. This chapter aims to present the student some ontology design solutions used for recurrent use cases, called ontology design patterns. Examples are provided for structural, correspondence, content, and presentation ontology design patterns. The aim is to encourage students to build ontologies by composing already verified design solutions.

Chapter 8 - Debugging Ontology. This chapter introduces the technical instrumentation required to identify and solve errors in an ontology. First, common inconsistency errors are exemplified. Then, the satisfiability is analysed from different perspectives: concepts satisfiability, role satisfiability and TBox satisfiability. The learner is advised how to identify cycling definitions. Having an image on the errors in the current ontology, the discourse continues with how to repair these errors. The strategy enacted in the chapter is first to obtain explanation about the error source and then to repair those definitions with the highest contribution to the error.

Chapter 9 - Ontology Evaluation. This chapter provides the conceptual instrumentation needed to evaluate an ontology. The ontology evaluation metrics detailed in this chapter deals with both structural evaluation an semantic evaluation. The MiniLisp and LRacer API are introduced to support the student in his task to develop various evaluation metrics, integrated in the RacerPro environment. Also, the chapter makes the student aware of some worst practices that occur during ontology engineering.

Chapter 10 - Documenting Ontologies. The chapter has two objectives. The first objective is to introduce the most common annotation properties used to document the ontology. The second one is to introduce the technical instrumentation which automatically generates ontology documentation. We focus on HTML and Latex format.

Chapter 11 - Engineering a Romanian Tourism Ontology. This chapter presents a large domain ontology for the Romanian tourism. The ontology was developed in the KRSS syntax in a modular way. After introducing the core ontology, the chapter details various modules that encapsulate knowledge on: accommodation types and their facilities, touristic activities, points of interest or eating and drinking resorts. For populating the ontology several sources were linked like Foursquare, Open StreeMap or the Open Linked Data for Romania.

Chapter 12 - Engineering a Vehicular Network Ontology. To develop the Vehicular Network (VANET) ontology, we follow the methodology detailed throughout the previous chapters and we also enact ontology design patterns. The engineering steps presented in this chapter are: i) defining competency questions, ii) reusing other ontologies, iii) defining main concepts and roles, iv) populating the ontology, and v) ontology debugging and evaluation.

Each chapter proposes a set of exercises. Sources in the KRSS syntax or guidelines for solving the exercises are provided at the end of each chapter.

I would like to thank to Prof. Ioan Alfred Letia and Prof. Alin Suciu, who carefully reviewed the manuscript, and whose valuable comments were used to improve the current version of the work. I kindly ask the reader to be indulgent when facing typos or confusing explanation, with the promise to improve the manual in a follow-up edition.

List of Figures

List of Tables

Part I

Active Learning of Ontology Engineering

Chapter 1

Ontologies in the KRSS syntax

This chapter introduces the technical instrumentation used throughout the paper. The KRSS syntax for developing ontologies is introduced by means of two basic ontologies: family and pizza. The RacerPro server and the RacerPorter client are described as the main technology used to support ontology engineer.

1.1 Short Trip to RacerPro and RacerPorter

RacerPro (Renamed ABox and Concept Expression Reasoner Professional) is a server able to reason on the ontologies loaded on it. RacerPorter is a Graphical User Interface running as a client for RacerPro. When RacerPorter is run, it automatically starts RacerPro and connects to it.

RacerPro functions and macros are available at http://localhost:8080/allfunctions.html. Most of them are described in the RacerPro reference manual [32]. The user guide provided by the developers of RacerPro is [33]. RacerPorter provides a read-eval-print loop for interactively querying the RacerPro server. An example session for querying the *shell* follows:

```
? (get-racer-version)
> "2.0"
? (get-server-timeout)
> NIL
? (set-server-timeout 5)
> :OKAY
? (get-server-timeout)
> 5
```

The prompter `?` waits for a command, while the answer appears after the `>` sign. Depending on your local network configuration, RacerPro requires the proxy setting to access ontologies specified by an uri.

```
? (get-proxy-server)
> NIL
? (set-proxy-server "http://192.168.1.2:3128")
> "http://192.168.1.2:3128"
? (get-proxy-server)
> "http://192.168.1.2:3128"
```

Now, you can load ontologies from online sources:

```
? (OWLAPI-readOntology
```

```
        "http://130.88.198.11/co-ode-files/ontologies/pizza.owl")
> NIL
```

In the following, we analyse the Pizza ontology by means of some RacerPro commands.

To list the prefixes defined in the current ontolog, one can use:

```
? (get-prefixes)
> (("defaultnamespace"
        "http://www.co-ode.org/ontologies/pizza/pizza.owl#")
   (NIL "http://www.co-ode.org/ontologies/pizza/pizza.owl#")
   ("swrl" "http://www.w3.org/2003/11/swrl#")
   ("owl" "http://www.w3.org/2002/07/owl#")
   ("owl2" "http://www.w3.org/2002/07/owl#")
   ("xsd" "http://www.w3.org/2001/XMLSchema#")
   ("rdfs" "http://www.w3.org/2000/01/rdf-schema#")
   ("rdf" "http://www.w3.org/1999/02/22-rdf-syntax-ns#"))
```

We continue by retrieving the individuals, atomic concept and roles formalised in the Pizza ontology.

```
? (all-individuals)
> (#!:America #!:Italy #!:Germany #!:France #!:England)
? (all-atomic-concepts)
> (TOP BOTTOM #!:American #!:TomatoTopping #!:PeperoniSausageTopping
        #!:MozzarellaTopping #!:NamedPizza #!:AnchoviesTopping
        #!:FishTopping #!:ArtichokeTopping #!:VegetableTopping #!:Mild
       ..........................................................)
? (all-roles)
> ( #!:isBaseOf #!:hasBase #!:isIngredientOf #!:hasIngredient
        #!:isToppingOf #!:hasTopping #!:hasSpiciness
        ..........................................................)
```

Note that the `#!:` denotes the current namespace. The concepts are hierarchically organised in a taxonomy (see Fig. 1.1).

Similarly, the roles are organised into a role hierarchy (see Fig. 1.2).

The concepts and roles are defined into a terminological box (TBox), while individuals in an assertional box (ABox).

```
? (current-tbox)
> http://130.88.198.11/co-ode-files/ontologies/pizza.owl
? (current-abox)
> http://130.88.198.11/co-ode-files/ontologies/pizza.owl
```

A description of a specific concept in a given TBox is obtained with:

```
? (describe-concept #!:Food
   http://130.88.198.11/co-ode-files/ontologies/pizza.owl)
> (#!:Food
        :TOLD-PRIMITIVE-DEFINITION #!:DomainConcept
        :SYNONYMS (#!:Food)
        :PARENTS  ((#!:DomainConcept))
        :CHILDREN ((#!:PizzaBase) (#!:Pizza) (#!:PizzaTopping)))
```

Note that the loaded ontology is in `OWL` (Web Ontology Language) syntax. Throughout this paper we will use the KRSS syntax for representing ontologies. Conversion between these two languages is easily performed in RacerPro by specifying the syntax in which the knowledge base (KB) is saved:

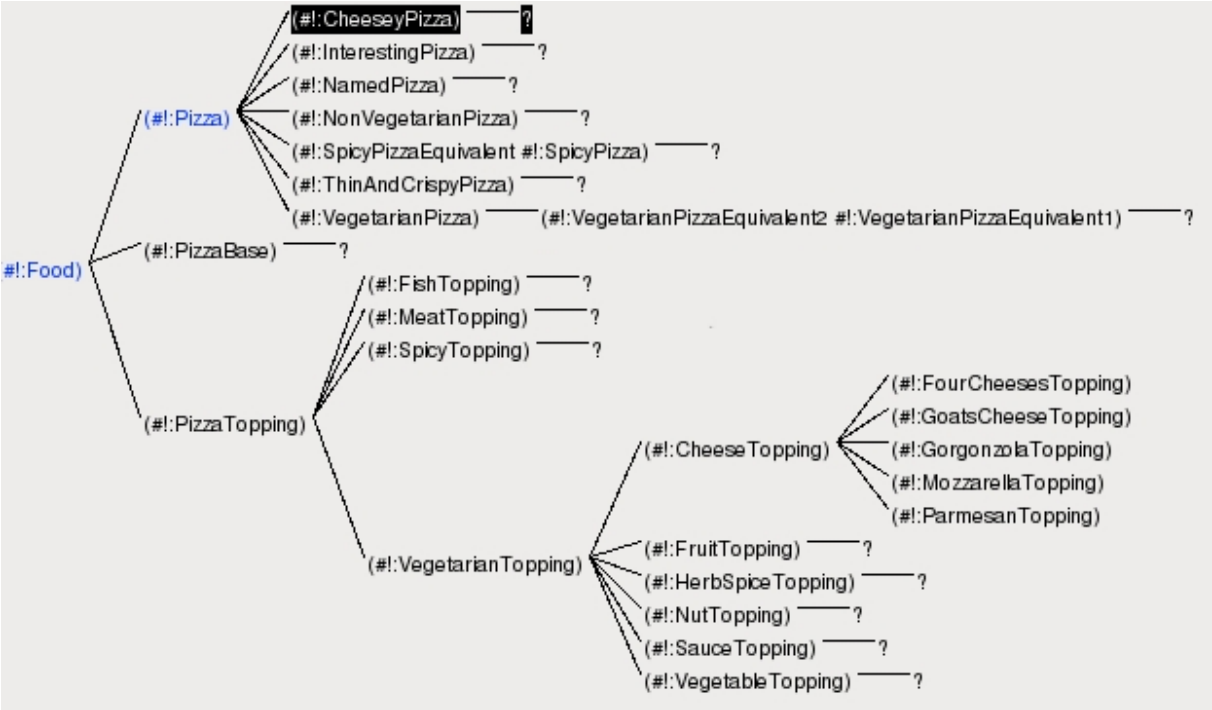

Figure 1.1: Taxonomy for the Pizza ontology.

Figure 1.2: Role hierarchy in the Pizza ontology.

```
? (save-kb "/home/kbs/pizza.racer" :syntax :racer)
> OKAY
```

The next section aims to introduce the KRSS syntax.

1.2 Description logic in the KRSS syntax

KRSS stands for Knowledge Representation System Specification [27]. KRSS uses Lisp-like syntax.

An ontology contains concepts, roles, and individuals (see Fig. 1.3). A concept (circle) is a set of individuals or instances (diamonds). Roles are binary relations (arrows) that link two individuals.

In the description logic (DL) version $\mathcal{ALC}$, concepts are built using the set of constructors formed by negation, conjunction, disjunction, value restriction, and existential restriction [2], as shown in Table 1.1. Here, C and D represent concept descriptions, while r is a role name. The semantics is defined based on an interpretation $I = (\Delta^I, \cdot^I)$, where the domain Δ^I of I contains a non-empty set of individuals, and the interpretation

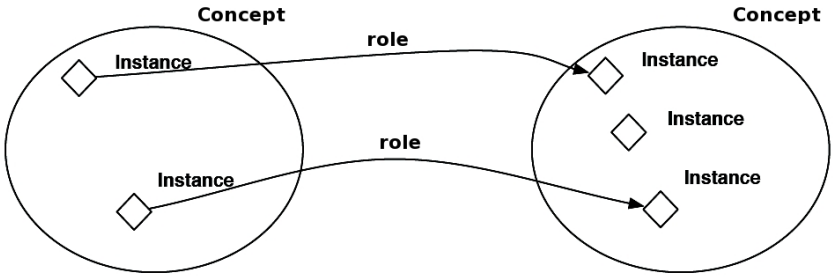

Figure 1.3: Elements in an ontology.

Constructor	*Syntax*	*Semantics*
negation	`(not C)`	$\Delta^I \setminus C^I$
conjunction	`(and C D)`	$C^I \cap D^I$
disjunction	`(or C D)`	$C^I \cup D^I$
existential restriction	`(some r C)`	$\{x \in \Delta^I \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$
value restriction	`(all r C)`	$\{x \in \Delta^I \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$
individual assertion	`(instance a C)`	$\{a\} \in C^I$
role assertion	`(related a b r)`	$(a^I, b^I) \in r^I$

Table 1.1: KRSS syntax and semantics of $\mathcal{ALC}$.

function $\cdot^I$ maps each concept name C to a set of individuals $C^I \in \Delta^I$ and each role r to a binary relation $r^I \in \Delta^I \times \Delta^I$. The last column of table 1.1 shows the extension of $\cdot^I$ for non-atomic concepts.

An ontology consists of terminologies (or TBoxes) and assertions (or ABoxes). A terminology *TBox* is a set of terminological axioms of the form `(equiv C D)` or `(implies C D)`.

In the TBox `Vanet` in Fig. 1.4, the vehicles are partitioned into private (belonging to individuals or private companies in line 2) and public (i.e. buses, police in lines 5-6). The axiom 8 specifies that a `PublicVehicle` should belong only to public agencies. In line 10, road side units can belong to the government or private service operators.

An assertional box *ABox* is a finite set of concept assertions `(instance a C)` or role assertions `(related a b r)`, where `C` designates a concept, `r` a role, and `a` and `b` are two individuals. Usually, the unique name assumption holds within the same *ABox*.

The assertional box `vanet-cluj` makes use of the terminologies in the `Vanet` tbox (line

```
1  (in-tbox Vanet)
2  (define-primitive-role belongsTo :domain Vehicle
3              :range (or Individual Company PublicAgency))
4  (implies PrivateVehicle Vehicle)
5  (implies PublicVehicle Vehicle)
6  (implies Bus PublicVehicle)
7  (implies Police PublicVehicle)
8  (implies PublicVehicle (all belongsTo PublicAgency))
9  (implies LocalTransportAgengy PublicAgency)
10 (implies RoadSideUnit (some belongsTo (or PublicAgency PrivateSerOp)))
```

Figure 1.4: Modeling knowledge in the vehicular network domain.

```
1 (in-abox vanet-cluj Vanet)
2 (instance b1 Bus)
3 (instance lta-cluj LocalTransportAgengy)
4 (instance rsu1 RoadSideUnit)
5 (related b1 lta-cluj belongsTo)
6 (related rsu1 lta-cluj belongsTo)
```

Figure 1.5: Modeling assertions in vehicular network domain.

Bus	$\sqsubseteq$ PublicVehicle
LocalTransportAgengy	$\sqsubseteq$ PublicAgency
Police	$\sqsubseteq$ PublicVehicle
PrivateVehicle	$\sqsubseteq$ Vehicle
PiblicVehicle	$\sqsubseteq \forall$belongsTo.PublicAgency
PublicVehicle	$\sqsubseteq$ Vehicle
RoadSideInit	$\sqsubseteq \exists$belongsTo.(PrivateSerOp $\sqcup$ PublicAgency)
$\exists$belongsTo.$\top$	$\sqsubseteq$ Vehicle
$\top$	$\sqsubseteq \forall$belongsTo.(Company $\sqcup$ Individual $\sqcup$ PublicAgency)

Figure 1.6: Representing the vehicular network TBox in DL.

1 in Fig. 1.5). The bus `b1` (line 2) belongs to the local transportation agency `lta-cluj` (line 3). Similarly, the road side unit `rsu1` (line 4) operates under the same public agency `lta-cluj` (line 6).

The corresponding representation of the vehicular network ontology in the DL syntax is illustrated in Fig. 1.6 and Fig. 1.7.

b1 : Bus
belongsTo(b1, Lta-cluj)
lta-cluj : LocalTransportationAgency
rsu1 : RoadSideUnit
belongsTo(rsu1, lta-cluj)

Figure 1.7: Representing the vehicular network ABox in DL.

A concept C is satisfied if there exists an interpretation I such that $C^I \neq \emptyset$. The concept D subsumes the concept C, represented by `(implies C D)` if $C^I \subseteq D^I$ for all interpretations I. Constraints on concepts (i.e. `disjoint`) or on roles (`domain`, `range` of a role, `inverse` roles, or `transitive` properties) can be specified in more expressive description logics. We provide only some basic terminologies of description logics. For a detailed explanation about families of description logics, the reader is referred to [2].

1.3 Family ontology explained

The family ontology is provided by the RacerPro developers to introduce the KRSS syntax and the basic query capabilities of RacerPro. The following explanations aim to complement the description of family ontology from [33].

The macro `in-knowledge-base` initialises the TBox `family` and the ABox `smith-family`:

```
(in-knowledge-base family smith-family)
```

After initialisation, the macro `signature` can follow to define the signature of the knowledge-base. Here, the signature enumerates concepts, roles, and individuals used in the ontology. Note that after ':' keywords of the RacerPro language follows, like `atomic-concepts`, `roles`, `individuals`, `parent`, `transitive` and `feature`. For instance, `:parent` says that role `has-child` is a subrole of role `has-descendant`. It means that, if individual `alice` has as child instance `betty`, RacerPro is able to infer that `alice` has descendant `betty`. The keyword `:transitive` associated to role `has-descendant` defines this role as a transitive one. Hence, if `alice` has descendant `betty` and `betty` has descendant `doris`, the system deduces that `alice` also has descendant `doris`.

```
(signature
    :atomic-concepts (person human female male woman man parent mother)
    :roles           ((has-child      :parent has-descendant)
                      (has-descendant :transitive t)
                      (has-sibling)
                      (has-gender :feature t))
    :individuals     (alice betty charles doris eve))
```

TBox. The family TBox starts by defining domain and range restrictions for roles. For specifying the range of role `has-child`, the expression

```
(implies *top* (all has-child person))
```

says that all the assertions for roles `has-child` point towards the concept `person`. Here, the built-in concept `*top*` represents the most general concept of each TBox. Opposite, the built-in concept `*bottom*` is the name of the incoherent concept (or empty set).

For specifying the domain of the role `has-child`, the sentence

```
(implies (some has-child *top*) parent)
```

states that individuals having the role `has-child` are of type `parent`. Thus, the role `has-child` starts from the concept `parent` and points toward the concept `person`.

The TBox also contains axioms for defining concepts. A person is a human which has gender female or male:

```
(implies person (and human (some has-gender (or female male))))
```

where female and male are disjoint, given by:

```
(disjoint female male)
```

A parent is defined as a person which has at least one child of type person:

```
(equivalent parent (and person (some has-child person)))
```

The macro `(implies A B)` gives the necessary conditions `A` for the concept `B`. The macro `equivalent` states the equality between two concepts.

ABox. Specific information about individuals is provided in an ABox. For instance, the individual `alice` is an instance of the concept `mother`, written as:

```
 (instance alice mother)
```

To specify the child `betty` of `alice`, the `related` macro is used:

```
(related alice betty has-child)
```

Q-box. Query boxes are used to group queries used to retrieve infomation from an ontology. Basic queries presented in the family ontologies are described in the next paragraphs.

The query `concept-ancestors` gets all atomic concepts of a TBox, which are more general (subsumed) by a specific concept. The query:

```
(concept-ancestors mother)
```

returns a list of more general concepts than `mother`, like `(woman parent human person *top*)`.

To obtain more specific concepts than a given one, the macro `concept-descendands` can be used. Executing the query:

```
(concept-descendants man)
```

the system enumerates concepts included in the concept `man`, i.e., `brother` or `father`.

The macro `(individual-instance i C)` checks if individual `i` is an instance of concept `C`. Checking that `doris` is a `woman` returns true (`T`):

```
? (individual-instance? doris woman)
> T
```

To obtain a list of all individuals that are instances of a concept, the `concept-instances` macro is used. With:

```
(concept-instances sister)
```

individuals like `doris` and `eve` are obtained because they are sisters in the current ABox.

1.4 Exercises

Exercise 1.1 (Family ontology)**.** *Load the 'family' ontology in RacerPorter and execute the following queries in the Shell tab:*

```
(concept-subsumes? brother uncle)
(concept-ancestors mother)
(concept-descendants man)
(all-transitive-roles)
(individual-instance? doris woman)
(individual-types eve)
(individual-fillers alice has-descendant)
(individual-direct-types eve)
(concept-instances sister)
```

Exercise 1.2 (RacerPro)**.** *Test the following functions which correspond to RacerPorter tabs:*

```
(all-aboxes)
(all-tboxes)
(all-atomic-concepts)
```

```
(all-roles)
(all-individuals)
(all-queries)
(taxonomy)
```

Exercise 1.3 (RacerPro). *Test the following functions for TBox management:*

```
(get-tbox-signature)
(current-tbox)
(associated-aboxes)
```

Consult the RacerPro manual [23] for their description.

Exercise 1.4 (RacerPro). *Test the following functions for ABox management:*

```
(get-abox-signature)
(get-kbbox-signature)
(current-abox)
(get-abox-version)
(associated-tbox)
```

Consult the RacerPro manual [23] for their description.

Exercise 1.5 (KRSS syntax). *Load an ontology in OWL format, for instance* `people-pets.owl`. *Convert the ontology in KRSS syntax. Analyse the code. Solution*

Exercise 1.6 (KRSS syntax). *Convert the family ontology in OWL format. Then convert back the OWL file in KRSS syntax. Compare the initial* `family.racer` *file with the resulted one. Solution*

1.5 Solutions

Solution 1.1 (Exercise 1.5). *An OWL ontology can be loaded with:*

```
(OWLAPI-readOntology "/path/people-pets.owl")
```

See the function `owl-read-file` *and friends. For saving the ontology in KRSS syntax you can test:*

```
(save-tbox  "/path/people-pets.racer")
(save-abox  "/path/people-pets.racer")
(save-kb  "/path/people-pets.racer")
(save-kb "/path/people-pets.racer" :tbox people-pets
                                   :abox my-pets
                                   :syntax :krss)
```

Solution 1.2 (Exercise 1.6).

```
(racer-read-file "/path/family.racer.")
```

Save the ontology in OWL format with:

```
(save-kb "/path/people-pets.racer" :syntax :owl)
```

Chapter 2

Reusing Ontologies

The chapter aims to familiarise the learner with the current development of Semantic Web. The student is encouraged to explore upper level ontologies and domain ontologies by means of semantic search engines and various ontology repositories. The selected ontologies are converted in the KRSS syntax and analysed for possible reuse. Competency questions are introduced to narrow the scope of the ontology to be developed.

2.1 Determining the domain of the ontology

A solution to determine the scope of an ontology is to start by defining a list of *competency questions* (CQs) [36]. A competency question (CQ) is a typical query that an expert might want to submit to a an ontology from his expertise domain.

Competency questions are questions that an ontology should be able to answer in order to satisfy use cases. Thereby, CQs represent initial requirements and they can be used to validate the ontology. Having the role of a requirement, each CQs are written in natural language. For the validation task, the CQs are formalised in query languages such as SPARQL (Simple Protocol and RDF Query Language) [41] or nRQL (new Racer Query Language) [23].

As a first example, consider the task to build an ontology for the domain of Human Resource Management in IT companies. Possible competency questions appear in Table 2.1. For instance the CQ_{13} is represented in nRQL in listing 2.1.

Listing 2.1: Formalising CQ_{13}

```
1  (retrieve (?cv) (and (?x Candidate)
2                       (?x (>= hasJavaExperience 2))
3                       (?x ?cv hasCV)))
```

A second exemple, a list of competency questions for the tourism domain is exemplified in Table 2.2.

As a third example, consider the task to develop an ontology for the vehicular network domain. Table 2.3 sketches a list of competency question for the vehicular domain.

The domain of the ontology is also narrowed by defining *usage scenarios* for the ontology. An usage scenario for the tourism domain would be: *"Many young tourists will visit Cluj-Napoca, the Youth European Capital in 2015. With an expected average stay of 4 days, they want to see as much and variate as possible"*.

RacePro provides support for querying an ontology with both SPARQL and nRQL. Queries can be included in a query box (QBox), enabled with:

CQ_1	Which are the candidates with Java background?
CQ_2	Which are the candidates with OOP experience?
CQ_3	Which are strong points for selecting candidate X?
CQ_4	Which are weak points of candidate X?
CQ_5	Which is the age/education/etc of candidate X?
CQ_6	Which are current criteria for recruitment?
CQ_7	Which are open positions in the company?
CQ_8	Which is the ideal candidate?
CQ_9	What relationships exist between candidate X and current employees?
CQ_{10}	Which is the CV of candidate X?
CQ_{11}	What are the reasons of candidate X to apply for position Y?
CQ_{12}	What features can be extracted from the motivational letter?
CQ_{13}	Which are CVs of candidates with at least 2 years Java experience?

Table 2.1: Sample of competency questions for a human resource management ontology.

CQ_1	What services are included in a specific accommodation?
CQ_2	What time is check-in and check-out for a specific accommodation?
CQ_3	What places to eat and drink are there within a given distance?
CQ_4	What points of interest are around the accommodation?
CQ_5	What activities can you do around the accommodation?
CQ_6	Which are the travelling options around a specific point of interest?

Table 2.2: Sample of competency questions for a tourism ontology.

```
1 (enable-query-repository)
2 (show-qbox-for-abox)
3 (show-current-qbox)
```

2.2 Ontology classification

Ontologies can be classified according to their level of genericity using the classification model from [22] in four categories: *foundational*, *domain*, *task*, and *application*.

Foundational ontologies are ontologies that have a large scope and are domain independent. They include very general and basic concepts like `Object`, `Event`, `Quality`, `Agent`. Consequently, they are highly reusable in different modeling scenarios. Usually, these general ontologies are well founded in various philosophical theories [4]. Examples of upper-level ontologies are:

- SUMO[1] (Suggested Upper Merged Ontology): As one of the largest available ontologies, it contains 20,000 terms and 60,000 axioms.
- DOLCE[2] (Descriptive Ontology for Linguistic and Cognitive Engineering): At its top level, DOLCE ontology distinguishes between `Endurant`, `Perdurant`, `Quality`, and `Abstract`. The main relation between `Endurants` (i.e., `Object` or `Substance`)

[1] http://www.ontologyportal.org/
[2] http://www.loa.istc.cnr.it/old/DOLCE.html

CQ_1	Is there an accident on street X?
CQ_2	How dense is the traffic on street x?
CQ_3	What is the distance to the car in front of my car?
CQ_4	What is the model of the car behind me?
CQ_5	How many cars in my neighborhood have a velocity greater than the legal one?
CQ_6	Is there a radar on street x?

Table 2.3: Sample of competency questions for a vehicular networks ontology.

and `Perdurants` (i.e., `Event` or `Process`) is that of participation: an `Endurant` 'lives' in time by participating in a `Perdurant`. `Quality` includes concept that can be perceived or measured. It contains around 100 concepts and 100 axioms [17].

- Cyc[3] - formalises facts, rules of thumb and heuristics for reasoning about the objects and events of everyday life [8]. The ontology contains more than 300,000 concepts, 15.000 relations, and 3,000,000 assertions divided into microtheories.

- BFO (Basic Formal Ontology[4]) - consists of 36 classes divided in two sub-ontologies. The first one (SNAP) contains all entities existing at a time. The second one (SPAN) includes all processes unfolding through time [20].

- SIOC (Semantically-Interlinked Online Communities Project) focuses on social network and activities by providing methods for interconnecting blogs, forums and mailing list [5].

Domain ontologies formalise the vocabulary related to a generic domain (like geographical, vehicular). Domain ontologies are the basic ontologies for real-world applications. As an example, the music ontology (http://musicontology.com) encapsulates information related to the music industry. Being a business-oriented ontology, it focuses on live events, albums, artists, tracks.

Task ontologies describe generic task or activity (selling, diagnosis). For instance, *Good Relations* ontology (http://purl.org/goodrelations/) aims to be a Web vocabulary for e-commerce, focusing on describing products sold online.

Application ontologies specialise both domain and task ontologies. In RacerPro you can load multiple ontologies into a single knowledge base, by specifying the same knowledge base in the parameter `:kb-name`.

```
1 (owl-read-document "http://zeitkunst.org/bibtex/0.2/bibtex.owl"
2                    :kb-name book-provenance)
3 (owl-read-document "http://inference-web.org/2.0/pml-provenance.owl"
4                    :kb-name book-provenance  :init nil)
```

2.3 Knowledge search over the Semantic Web

To search knowledge in the Semantic Web, two starting options are: exploiting *semantic search engines* or browsing *ontology repositories*.

[3]http://www.cyc.com/
[4]http://www.ifomis.org/bfo/

Semantic search engines

- Sindice[5] - The Semantic Web Index offers both a user interface and an WebAPI to allow Semantic Web agents to retrieve RDF (Resource Description Framework) or OWL files [39]. The focus is on scalability to handle large quantities of data.
- Watson[6] - is a gateway to the Semantic Web that provides a set of APIs for finding, exploring and querying semantic data and ontologies [11]. To facilitate ontology reusing and filtering, Watson retrieves metadata about an ontology (language, size, labels, comments, imported ontologies) and various ontology metrics (depth of the hierarchy, density of an entity).
- Swoogle - Semantic Web Google[7] - is a crawler-based indexing, retrieval and ranking system for ontologies on the Web [12].

Ontology Repositories.

Ontology libraries are usually designed for specific types of applications [25]. Therefore, they provide more specific information compared to semantic web search engines. Some popular repositories are:

- Knoodl (http://knoodl.com/ui/home.html)
- Bioportal (http://bioportal.bioontology.org/) contains over 300 biomedical ontologies with tools for working with these ontologies.
- Deri vocabularies (http://vocab.deri.ie/)
- TONES (http://owl.cs.manchester.ac.uk/repository/)
- OntoHub (http://ontohub.org)

The following commands can be used in RacerPro to include an ontology:

- `include-kb` loads a file in KRSS syntax. The function is used for partitioning a Tbox or an Abox into several files.
- `kb-ontologies` retrieves all the ontologies imported by the ontology given as parameter.

2.4 Exercises

Exercise 2.1 (Ontology classification)**.** *Load the 'Good Relation' application ontology in RacerPro. Identify how many concepts, roles, attributes, and individuals it contains. Solution*

Exercise 2.2 (Reusing ontologies)**.** *Browse several ontology repositories.*

1. *Identify and save locally ontologies related to the tourism domain. Be aware of the size of the saved ontologies.*

[5] http://sindice.com/
[6] http://watson.kmi.open.ac.u
[7] swoogle.umbc.edu/

2. *Select an ontology that you consider useful to be re-used. Translate this ontology in KRSS syntax.*

3. *Load the ontology in RacerPorter and analyse it.*

Solution

Exercise 2.3 (Reusing ontologies)**.** *Assume you are asked to develop an ontology about*

1. *Human diseases*

2. *Tourism*

3. *Software*

4. *Universe*

5. *Dynosaurus*

6. *Human Resource Management*

Which (if any) foundational ontology would you choose for each one? Why?

Exercise 2.4 (Reusing ontologies)**.** *Imagine that your final ontology will be a successful one. Identify online repositories where you can store and advertise your ontology. Identify tools that can be used to launch your own ontology repository.*

Exercise 2.5 (Competency questions)**.** *Read some papers in domain of the ontology you chosen to implement. Write 10 competency questions inspired from what you have found. Give justification/evidence/reference that CQs are relevant for the domain. Write a very short .tex document containing these CQs and references to the papers you have read, included as .bib entries. Solution*

Exercise 2.6 (Competency questions)**.** *Write CQs for the 'Functional Programming' domain. Identify main terms from the CQs defined by you. Nouns are the basis for defining concepts, while verbs for defining roles. For each identified term, build a list of more general and a list of more specific terms. Solution*

Exercise 2.7 (Competency questions)**.** *Consider the task to build an ontology for the software domain. Write CQs and group them based on the following aspects: function, data, license, etc. Solution*

2.5 Solutions

Solution 2.1 (Exercise 2.1)**.** *Good Relations is a relatively small application ontology containing 27 concepts, 43 roles, 37 attributes and 43 individuals.*

Solution 2.2 (Exercise 2.2)**.** *You can use OWLAPI or online translators like http://owl.cs.manchester.ac.uk/converter/.*

Solution 2.3 (Exercise 2.5)**.** *Consult the following online libraries: ScienceDirect, Springer, IEEEXplore. You can use latex editors like Kile (http://kile.sourceforge.net/).*

Solution 2.4 (Exercise 2.3)**.** *For instance, an ontology for Software that exploits the DOLCE ontology is described in [38].*

Solution 2.5 (Exercise 2.6)**.** *For example, the term FunctionalProgramming has more general terms:*

```
(implies FunctionalPrograming DeclaritiveProgramming)
(implies FunctionalProgramming Programming Paradigm)
(implies FunctionalProgramming ComputerScience)
```

and more specific terms:

```
(implies PureFunctionalProgramming FunctionalProgramming}
(implies LambdaCalculus FunctionalProgramming)
```

You may consult dictionaries like WordNet (http://wordnet.princeton.edu/).

Solution 2.6 (Exercise 2.7)**.** *A list of CQ for the software domain is defined by the Software Ontology Project (SWOT). Sample of these competency questions taken from http://softwareontology.wordpress.com appears below:*

Function	*What is the algorithm used to process this data?*
	What are the primary inputs and outputs?
	Is this software available as a web service?
Data	*What software can read a .tex file?*
	Is this software compatible with x?
License	*What license and what is permissiveness?*
	Who owns the copyright?
	What is the licensing history?

Note that for large domains, the CQs are grouped in cathegories (in this case Function, Data and Licence).

Chapter 3

Defining Concepts

This chapter starts by introducing the primitives used to formalised concepts in an ontology. The learners are familiarised with the subsumption relation between concepts. Then, the chapter presents queries available to retrieve information about the concepts in a knowledge base. The last part exemplifies how to handle terminological boxes, as the main building blocks of grouping axioms in a large ontology.

3.1 Concept axioms

Generic concepts represent classes, divided into defined and primitive concepts. A *defined concept* specifies the necessary and sufficient conditions for a given entity to be considered an instance of the class corresponding to the concept. A *primitive concept* specifies only the necessary conditions for a given entity to be classified as an instance of the class corresponding to the concept. Thus, a defined concept is given by a complete definition, and a primitive concept by an incomplete definition.

The most general concept of each TBox is the top concept ($\top$) denoted by `*top*`. The bottom concept ($\bot$), denoted by `*bottom*` represents the incoherent concept. These two built-in concepts are elements of every TBox.

Terminological axioms have the form $C \sqsubseteq D$ or $C \equiv D$. Axioms of the first kind are called *inclusions*, while axioms of the second kind are called *equalities*. Several types of concept axioms can be used to specify concepts:

General concept inclusion (GCI) states the subsumption relation ($C_1 \sqsubseteq C_2$) between two concepts, given by `(implies` C_1 C_2`)` in KRSS syntax. C_2 defines necessary conditions for C_1:

```
1 (implies Grandfather (and Parent Male))
```

Concept equations are used to state the equivalence ($C_1 \equiv C_2$) between two concepts, given by `(equivalent` C_1 C_2`)`:

```
1 (equivalent Grandfather (and Father (some has-child Parent))
```

Only one definition for a concept (or role) is allowed.

Concept disjointness axioms indicate that concepts do not have individuals in common ($C_1 \sqsubseteq C_2$), defined in RacerPro with `(disjoint` C_1`,...,` C_2`)`:

```
1   (disjoint Friday Sunday Monday)
2   (implies Book (not Journal))
3 ? (concept-disjoint? Book Journal)
```

```
> T
? (implies (and Apple Pear) *top*)
? (concept-disjoint? Apple Pear)
> T
```

```
  (implies Book Publication)
  (equivalent NonBook (and Publication (not Book))
? (concept-disjoint? Book NonBook)
> T
? (concept-equivalent Publication (or Book (not Book))
> T
```

In this case, RacerPro can deduce both the disjointness of `Book` and `NonBook`, and the fact that `Publication` is the union of `Book` and `NonBook`, without having stated anything explicitly about either.

RacerPro syntax allows: i) cyclic axioms; ii) forward references to concepts which will be introduced; iii) in case a concept is described by several axioms, following axioms are added and they do not overwrite the previous axioms [33].

The subsumption axiom $C \sqsubseteq D$ follows from a TBox if and only if the complex concept $C \sqcap \neg D$ is unsatisfiable in that TBox: $C \sqsubseteq D \; iff \; C \sqcap \neg D \sqsubseteq \bot$. Two concepts C and D are disjoint if and only if the complex concept $C \sqcap \neg D$ is unsatisfiable: `(disjoint C D)` iff $C \sqcap D \sqsubseteq \bot$.

3.2 Queries for concepts terms

To test whether a concept is satisfiable with respect to a TBox, the `concept-satisfiable?` macro can be used. If no TBox is specified the current-tbox is assumed. The macro returns `nil` when the given concept is unsatisfiable, as in:

```
? (concept-satisfiable? (some r A))
> t
  (equivalent A (and B C))
  (disjoint B C)
? (concept-satisfiable? A)
> NIL
? (concept-satisfiable? (and (all r A) (some r (not A))))
> NIL
```

To check if a concepts subsumes another concept, the macro `concept-subsumes?` C_1 C_2 can be enacted: The macro returns `true` is C_1 subsumes C_2 ($C_2 \sqsubseteq C_1$) and `nil` otherwise.

```
  (equivalent Parent (and Person (some has-child Person)))
  (equivalent Grandparent (and Person (some has-child Parent)))
? (concept-subsumes? Parent Grandparent)
> T
```

The macro `(concept-equivalent? C1 C2)` checks if concepts `C1` and `C2` are equivalent in the TBox `tbox`. If the TBox is not specified, the current TBox is assumed:

```
 (concept-equivalent? (some r A) (not (all r (not A))))
```

The macro `(concept-disjoint?` C_1 C_2`)` tests if the two concepts C_1 and C_2 are disjoint. Recall that, no individual can be an instance of two disjoint concepts.

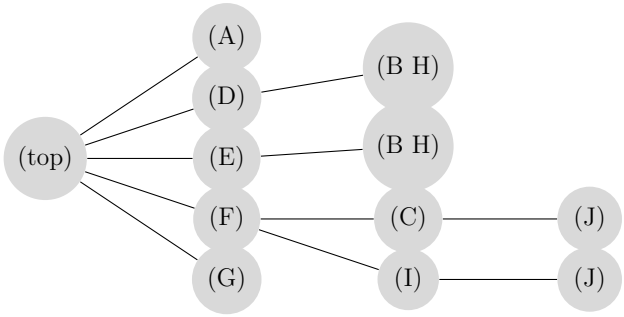

Figure 3.1: The taxonomy of the TBox `tbox-retrieval`.

```
(concept-disjoint? A (not A))
```

3.3 TBox retrieval

A knowledge base may include several TBoxes. The macro `(all-tboxes)` returns all loaded tboxes. Only one TBox is active, whose name is obtained by the command `(current-tbox)`.

The subsumption relationships between concepts form the taxonomy. The taxonomy of a specific TBox is obtained with the command `(taxonomy)`. Assume the following TBox.

```
  (full-reset)
  (in-tbox tbox-retrieval)
  (implies A (or B C))
  (implies B (and D E))
  (implies C (and F (some r G)))
  (equivalent I (and F (all r J)))
  (equivalent J (and C I))
  (equivalent H B)
? (all-tboxes)
> (OWLAPI-KB default tbox-retrieval))
? (all-atomic-concepts)
> (top bottom G C F B E A H D)
? (taxonomy)
> ((top nil (A D E F G)) (A (top) (bottom)) ((B H) (D E) (bottom))
   (C (F) (J)) (D (top) ((B H))) (E (top) ((B H))) (F (top) (C I))
   (G (top) (bottom)) (I (F) (J)) (J (C I) (bottom))
   (bottom (A (B H) G J) nil))
```

The function `(full-reset)` deletes all TBoxes, ABoxes, queries and rules loaded in the RacerPro memory. The tree representation of the above nested list appears in Fig. 3.1.

Note that concepts `H` and `E` are equivalent. This can be checked with:

```
? (concept-synonyms B)
> (B H)
```

This is the only pair of equivalent concepts in the current TBox, verified by:

```
? (all-equivalent-concepts)
> ((*top* top) (*bottom* bottom) (G) (C) (J) (F)
   (B H) (I) (E) (A) (B H) (D))
```

Note that the unsatisfiable concept $\bot$ is denoted by two synonyms: `*bottom*` and `bottom`. Similarly, the most general concept $\top$ is expressed by two equivalent concepts: `*top*` and `top`.

The macro `concept-descendants` can be used to list all atomic concepts of a TBox, which are subsumed by the specified concept. In Fig. 3.1, the descendants of `F` are: `I`, `C` and `bottom`, verified by:

```
1 ? (concept-descendands F)
2 > ((*bottom* bottom) (J) (I) (C))
```

The macro `concept-ancestors` lists all atomic concepts which are subsuming the specified concept. For the ascensors of `J` we obtain:

```
1 ? (concept-ancestors J)
2 > ((*top* top) (F) (I) (C))
```

To get only direct subsumers of a concept in the TBox, the command `concept-children` is used. The children of `F` are:

```
1 ? (concept-children F)
2 > ((I) (C))
```

Similarly, the macro `concept-parents` returns only direct subsumers of the specified concept. The parents of concept `J` in the current TBox are `I` and `C`:

```
1 ? (concept-parents J)
2 > ((I) (C))
```

The function `classify-tbox` classifies the whole TBox. This function needs to be executed before queries can be posed. The definition of a concept is obtained with `(get-concept-definition)`, while its complete description with `describe-concept` The description of the concept includes its definition, synonyms, a list of parents and children concepts.

Least common subsumer. The *least common subsumer* (lcs) of concepts C_1,...C_n w.r.t. a TBox T is the most specific concept description D that subsumes C_1,...C_n w.r.t. T. Formally:

- D is a common subsumer: $C_i \sqsubseteq D$)
- D is least: if a concept E satisfies ($C_i \sqsubseteq E$) for i=1,..n then $D \sqsubseteq E$

```
1   (equivalent fifhteen (and five three))
2   (equivalent ten (and five two))
3 ? (lcs-unfold ten fiftheen)
4 > five
```

3.4 Exercises

Exercise 3.1 (Built-in concepts)**.** *Check that, after initialisation, the current TBox includes only the concepts `*top*` and `*bottom*`. Solution*

Exercise 3.2 (Concept subsumption)**.** *Determine the parents and children of a given concept. Solution*

Exercise 3.3 (Equivalent concepts)**.** *For each of the following pairs, explain why the concepts are equivalent or not:*

(a)	*top*	*(not bottom)*
(b)	*A*	*(not (not A))*
(c)	*(or A B)*	*(not (and (not A) (not B)))*
(d)	*(not (and A B))*	*(or (not A) (not B))*
(e)	*(some r C)*	*(not (all r (not C)))*
(f)	*(not (all r C)*	*(some r (not C))*
(g)	*(and A (or B C))*	*(or (and A B) (and A C))*
(h)	*(or A (and B C))*	*(and (or A B) (or A C))*

Solution

Exercise 3.4 (Equivalent concepts)**.** *Check in RacerPro that concept intersection and union are commutative, associative and idempotent. Solution*

Exercise 3.5 (Equivalent concepts)**.** *Check in RacerPro that negation can be shifted past quantifiers. Solution*

Exercise 3.6 (Concept equivalence)**.** *Check in RacerPro that existential and universal quantification can be seen as a special case of number restrictions. Solution*

Exercise 3.7 (Concept equivalence)**.** *Show in RacerPro that the following concepts are not equivalent:*

(a)	*(some r (and C D))*	*(and (some r C) (some r D))*
(b)	*(and C (or D E))*	*(or (and C D) E)*
(c)	*(and (some r (one-of i)) (some r (one-of j)))*	*(at-most 2 (one-of i j))*

Solution

Exercise 3.8 (Modelling in DL)**.** *Express the following sentences in KRSS syntax:*

1. *All employees are persons.*
2. *A father is a male who has a child.*
3. *A parent is a mother or a father.*
4. *A grandmother is a mother who has a child who is a parent.*
5. *Only humans have children that are humans.*

Solution

Exercise 3.9 (Modelling in DL)**.** *Using the individuals* `john` *and* `cnets`*, concepts* `Course`*,* `Lecturer`*,* `MSc` *and* `BSc`*, and roles* `teaches` *and* `hasDegree`*, represent the following knowledge base as an ALC knowledge base* `K`*:*

- *Everybody who teaches a course must either have an MSc degree or be a lecturer.*
- *Every lecturer teaches some course.*
- *Every lecturer has a BSc degree.*
- *Everybody with an MSc degree has a BSc degree as well.*

- *John teaches the Computer Networks course.*

Solution

Exercise 3.10 (Modelling in DL)**.** *Represent the following sentences in KRSS syntax:*

(a) If somebody owns something, they care for it.

(b) Healthy beings are not dead.

(c) Every cat is dead or alive

(d) Briggite is a happy cat owner.

Decide whether the following propositions are true and give evidence:

(a) KB is satisfiable

(b) `(concept-subsumes? (one-of briggite) Alive)`

(c) `(concept-subsumes? (and Dead Alive) *top*)`

(d) `(concept-subsumes? Alive Healthy)`

Solution

Exercise 3.11 (Modelling in DL)**.** *Create a DL knowledge base that models the following facts:*

(a) Mammals are animals that give birth to live young.

(b) Dogs are carnivorous mammals.

(c) Elephants are herbivorous mammals.

(d) Carnivores eat meat.

(e) Vertebrate is any animal with a backbone.

(f) Every fish is an animal that lives in water.

(g) A bird is vertebrate that has wings, legs and lays eggs.

(h) Rezy is a bird that eats insects and seeds only.

(i) Tweety has wings, legs and give birth to live young.

Exercise 3.12 (Concept subsumption)**.** *For each of the following pairs, is the concept in the first column subsumed by the concept in the second column? Explain your answer.*

(a)	*(and (all r A) (all r B))*	*(all r (and A B))*
(b)	*(all r (and A B))*	*(and (all r A) (all r B))*
(c)	*(or (all r A) (all r B))*	*(all r (or A B))*
(d)	*(all r (or A B))*	*(or (all r A) (all r B))*
(e)	*(and (some r A) (some r B))*	*(some r (and A B))*
(f)	*(some r (and A B))*	*(and (some r A) (some r B))*
(g)	*(or (some r A) (some r B))*	*(some r (or A B))*
(h)	*(some r (or A B))*	*(or (some r A) (some r B))*

Solution

Exercise 3.13 (Concept subsumption). *Given the TBox:*

```
(equivalent C (and (some r (not B)) (not A)))
(equivalent D (some r B))
(euivalent E (and (not (some r A) (some r D))))
```

Is the concept `(and (not (some r A)) (some r C))` *subsummed by the concept* `(all r E)`*? Solution*

Exercise 3.14 (Concept subsumption). *Which of the following statements are true? Explain your answer.*

1. `(concept-subsumes? (some r A) (all r A))`
2. `(concept-subsumes? (all r A) (some r A))`
3. `(concept-subsumes? (some r A) (all r bottom))`
4. `(concept-subsumes? (all r A) (all r bottom))`
5. `(concept-subsumes? (all r A) (all r top))`

Exercise 3.15 (Concept subsumption). *Which of the following statements are true? Explain your answer.*

1. `(concept-subsumes? (at-least 3 hasSon Child) (at-least 5 hasSon Child))`
2. `(concept-subsumes? (some hasSon Child) (at-least 5 hasSon Child))`
3. `(concept-subsumes? (at-most 2 hasSon Child) (at-most 3 hasSon Child))`
4. `(concept-subsumes? (at-least 6 hasSon Child) (exactly 5 hasSon Child))`
5. `(concept-subsumes? (some hasSon Child) (exactly 5 hasSon Child))`

Exercise 3.16 (Concept subsumption). *Consider the following TBox.*

- *A car is produced by a car maker.*
- *Car maker is a manufacturer.*
- *A manufacturer produces products.*

Which of the following hold true?

1. *Car is a product.*
2. *Car is a car maker.*

Exercise 3.17 (Concept subsumption). *Consider*

```
(implies A (some r B))
(implies A (all r C))
```

Which of the following hold true?

1. `(concept-subsumes? (some r (and B C)) A)`

2. `(concept-subsumes? (all r (and B C)) A)`

Solution

Exercise 3.18 (TBox). *Consider the following ontology from [51]:*

```
(in-tbox Disease)
(define-primitive-role cont-in :parent comp-of)
(implies Pericardium (and Tissue (some cont-in Heart)))
(implies Pericarditis (and Inflammation (some has-loc Pericardium)))
(implies Inflammation (and Disease (some acts-on Tissue)))
(implies (and Disease (some has-loc (some comp-of Heart)))
         (and Heartdisease (some has-state NeedsTreatment)))
(define-concept HDSNT (and Heartdisease (some has-state NeedsTreatment)))
```

1. *Classify the ontology.*

2. *Check if* `Pericarditis` *is a heart disease needing treatment.*

Exercise 3.19 (Least common subsumer). *Compute yourself the output of the following commands and then execute them in RacerPro:*

1. `(lcs *top* *bottom*)`

2. `(lcs *bottom* *bottom*)`

3. `(lcs *bottom* A)`

4. `(lcs *top* A)`

Exercise 3.20 (Least common subsumer). *Compute lcs(C,D) with respect to each of the following Tboxes:*

1. `(implies D C)`

2. `(implies C D)`

3. `(same-as C A) (same-as C B)`

4. `(equivalent C (some r CC)), (equivalent D (some s DD))`

5. `(equivalent C (at-least 2 CC)), (equivalent D (at-least 3 DD))`

6. `(equivalent C (at-most 2 CC)), (equivalent D (at-most 3 DD))`

3.5 Solutions

Solution 3.1 (Exercise 3.1).

```
? (full-reset)
? (all-atomic-concepts)
```

Solution 3.2 (Exercise 3.2). *The* `(concept-children Concept)` *and* `(concept-parents Concept)` *should be used.*

Solution 3.3 (Exercise 3.3).

```
? (concept-equivalent? top (not bottom))
> t
? (concept-equivalent? A (not (not A)))
> t
? (concept-equivalent? (or A B) (not  (and (not A) (not B))))
> t
? (concept-equivalent? (not (and A B))  (or (not A) (not B)))
> t
? (concept-equivalent? (some r C)  (not (all r (not C)) ))
> t
? (concept-equivalent? (not (all r C))  (not (some r  (not C)) ))
> t
? (concept-equivalent? (not (all r C))  (some r  (not C)) )
> t
? (concept-equivalent? (and A (or B C))  (or (and A B) (and A C)))
> t
? (concept-equivalent? (or A (and B C))  (and (or A B) (or A C)))
> t
? (concept-equivalent? (all R B) (or (all R B) (some r (and a (not a)))))
> t
```

$\exists r.A \sqcap \neg A$ *is equivalent to the* `bottom` *concept.*

Solution 3.4 (Exercise 3.4).

Concept intersection:

Commutativity	`? (concept-equivalent? (and C D) (and D C))` `> T`
Associativity	`(concept-equivalent? (and (and C D) E) (and C (and D E)))` `> T`
Idempotency	`(concept-equivalent? (and C C) C)` `> T`

Concept union:

Commutativity	`(concept-equivalent? (or C D) (or D C))` `> T`
Associativity	`(concept-equivalent? (or (or C D) E) (or C (or D E)))` `> T`
Idempotency	`(concept-equivalent? (or C C) C)` `> T`

Solution 3.5 (Exercise 3.5).

```
? (concept-equivalent? (not (some r C)) (all r (not C)))
> t
? (concept-equivalent? (not (all r C) (some r (not C)))
> t
? (concept-equivalent? (not (at-most n r C)) (at-least (n+1) r C))
> t
? (concept-equivalent? (not (at-least (n+1) r C)) (at-most n r C))
> t
```

Solution 3.6 (Exercise 3.6)**.**

```
? (concept-equivalent? (at-least 0 r C) *top*)
> t
? (concept-equivalent? (at-least 1 r C) (some r C))
> t
? (concept-equivalent? (at-most 0 r C) (all r (not C)))
> t
```

Solution 3.7 (3.7)**.**

```
? (concept-equivalent? (some r (and C D)) (and (some r C) (some r D)))
> NIL
? (concept-equivalent? (and C (or D E)) (or (and C D) E))
> NIL
? (concept-equivalent? (and (some r (one-of i)) (some r (one-of j)))
                       (at-most 2 (one-of i j)))
> NIL
```

Solution 3.8 (Exercise 3.8)**.**

```
(implies Employee Person)
(equivalent Father (and Male (some has Child)))
(equivalent Parent (or Mother Father))
(equivalent Grandmother (and Mother (some has-child Parent)))
(implies Human (all has-child Human))
```

Solution 3.9 (Exercise 3.9)**.**

```
(implies (some teaches Course) (some has-degree (or MSc Lecturer)))
(implies Lecturer (some teaches Course))
(implies Lecturer (some has-degree BSc))
(implies (some has-degree MSc) (some has-degree BSc))

(related john cnets teaches)
(instance cnets Course)
```

Solution 3.10 (Exercise 3.10)**.**

```
  (role-implies owns caresFor)
  (implies Healthy (not Dead))
  (implies Cat (or Dead Alive))
  (implies HappyCatOwner (and (some owns Cat) (all caresFor Healthy)))
  (instance briggite HappyCatOwner)

? (concept-subsumes? (one-of briggite) Alive)
? (concept-subsumes? (and Dead Alive) *top*)
? (concept-subsumes? Alive Healthy)
```

Solution 3.11 (Exercise 3.12)**.**

```
? (concept-subsumes? (and (all r A) (all r B)) (all r (and A B)))
? (concept-subsumes? (all r (and A B)) (and (all r A) (all r B)) )
? (concept-subsumes? (or (all r A) (all r B)) (all r (or A B)))
? (concept-subsumes? (all r (or A B)) (or (all r A) (all r B)) )
? (concept-subsumes? (and (some r A) (some r B)) (some r (and A B)))
? (concept-subsumes? (some r (and A B)) (and (some r A) (some r B)) )
? (concept-subsumes? (or (some r A) (some r B)) (some r (or A B)))
? (concept-subsumes? (some r (or A B)) (or (some r A) (some r B)))
```

Solution 3.12 (Exercise 3.13)**.**

```
? (concept-subsumes? (all r E) (and (not (some r A)) (some r C)))
```

Solution 3.13 (Exercise 3.17)**.**

```
? (concept-subsumes? (some r (and B C)) A)
> T
? (concept-subsumes? (some r (and B C)) A)
> T
```

Chapter 4

Defining Roles

A role is a potential relation between the members of the class corresponding to a concept and other concepts. Roles are used to differentiate concepts by specifying the ways a concept differs from the intersection of its immediate super-concepts in the taxonomy. A role may be associated with:

i) restrictions on the possible values it may assume (i.e., domain, range);

ii) the number of possible associations (i.e., number restrictions, functional);

iii) relationships with other roles (i.e., super-roles, inverse roles);

iv) intrinsec properties (i.e., transitive, reflexive, symmetric).

4.1 Role axioms

Roles can be specified with the following statement:

```
(define-primitive-role RoleName  &key
                        (transitive nil) (feature nil) (symmetric nil)
                        (reflexive nil) (inverse nil) (domain nil)
                        (range nil) (parents nil))
```

If individuals `i` and `j` are related via a transitive role `r` and the individuals `j` and `k` are related by the same role, then `i` and `k` are also related via `r`. As an instance, the role `has-descendant` is transitive, defined as:

```
1 (define-primitive-role has-descendent :transitive t)
```

Thus, if `charles` has a descendent `henry`, and `henry` has a descendent `george`, the system deduces, based on the transitivity property, that `charles` has descendant `george`.

```
1   (define-primitive-role has-descendent :transitive t)
2   (related charles henry has-descendent)
3   (related henry george has-descendent)
4 ? (individuals-related? charles george has-descendent)
5 > t
```

Features (or attributes) restrict a role to be a functional role. Thus an individual can only have up to one filler for this role. Typical examples for features are `hasMother`, `marriedWith`, or `locatedInCountry`. The following definitions are equivalent:

```
(define-primitive-role has-best-friends :feature t
                                        :inverse best-friend-of
                                        :parent has-friends)
(define-primitive-attribute has-best-friends :inverse best-friend-of
                                             :parent has-friends)
```

Trying to relate an individual `i` with two distinct individuals `j` and `k` via a feature (attribute) leads to inconsistency, as in the following example:

```
  (set-unique-name-assumption t)
  (define-primitive-role has-husband :feature t)
  (related caty brad has-husband)
  (related caty jonny has-husband)
? (abox-consistent?)
> NIL
```

Here, the unique name assumption is activated to state that all individuals are distinct. In the current ABox, `Caty` is related with two distinct individuals via the feature `has-husband`. These assertions make the current ABox inconsistent.

In analogy to the top concept, there is a universal role `*top-object-role*`, which connects all individuals of the described domain.

```
  (instance i *top*)
  (instance j *top*)
? (individuals-related? i j *top-object-role*)
> t
```

Corresponding to the bottom concept, the empty role `*bottom-object-role*` means that "all things do not relate to anything through the empty role" [33]. Note that, any role can be defined to be empty with:

```
(implies top (not (some my-empty-role top))).
```

The `my-empty-role` role and the default empty role `*bottom-object-role*` automatically becomes synonyms, checked with:

```
  (implies top (not (some r top)))
? (role-synonyms r)
> (r *bottom-object-role*)
```

Similar to concept hierarchies, RacerPro allows role hierarchies by defining subrole-relationships between roles. If `r` is a parent role for role `s`, then the individuals related by `s` are also related by `r`.

```
  (define-primitive-role s :parent r)
  (related i j s)
? (individuals-related? i j r)
t
```

Consider the following role hierarchy:

```
(implies-role has-mother has-parent)
(implies-role has-fother has-parent)
(implies-role has-parent has-ancestor)
(role-equivalent empty *bottom-object-role*)
```

Roles associated with a concept are also applicable to all specializations of that concept. Thus, concepts inherit roles of the most general concepts to which they are connected.

```
1   (define-primitive-role has-child :domain Person :range Person)
2   (related marcel stefan has-child)
3 ? (concept-instances Person)
4 > (stefan marcel)
```

A role can be a conjunction of other roles, expressed as a subrole of several parents:

```
1   (define-primitive-role r :parents (s t))
2   (related i j r)
3 ? (individuals-related? i j s)
4 > t
5 ? (individuals-related? i j t)
6 > t
```

The macro `implies-role` can also be used to define subsumption relationships between roles $r \sqsubseteq s$:

```
1 (role-implies hasDaughter hasChild)
2 (role-implies hasSon hasChild)
```

One usage of `implies-role` is to compose roles. The DL expression $r_1 \circ ... \circ r_n \sqsubseteq r$ is translated in RacerPro as:

```
1 (implies-role (r1 ... rn) r)
```

```
1 (define-primitive-role has-child :inverse has-parent)
2 (define-primitive-role hasBorrowed :inverse lentTo)
```

A role is *symmetric* if it is equivalent to its own inverse, e.g., $married To \equiv married To^{-}$, $has Brother \equiv has Brother^{-}$.

```
1   (define-primitive-role hasBrother :symmetric t)
2   (related marcel adrian hasBrother)
3 ? (individuals-related? marcel adrian hasBrother)
4 > t
5 ? (individuals-related? adrian marcel hasBrother)
6 > t
```

A role is *asymmetric* if it is disjoint from its own inverse, e.g

```
1   (define-primitive-role has-child :inverse has-parent :asymmetric t)
2 ? (role-disjoint? has-child has-parent)
3 > t
```

A role is *reflexive* if

```
1   (define-primitive-role knows :reflexive t)
2   (instance a *top*)
3 ? (individuals-related? a a knows)
```

Table 4.1 give examples of roles constrained with the above mathematical properties.

Table 4.1: Properties of roles.

Property	Example	Formalisation
Transitive	hasAncestor	(related a b r) (related b c r) → (related a c r)
Symmetric	hasSpouse	(related a b r) → (related b a r)
Asymmetric	hasChild	(related a b r) → not (related b a r)
Reflexive	hasRelative	(related a a r) for all a
Irreflexive	hasParent	(related a a r) for all a
Functional	hasHusbund	(related a b r) (related a c r) → b=c

Table 4.2: Inherited role properties: R- primitive role, R^+: transitive role, F: feature.

Child-role Parent-role	R	R+	F
R	R	R	F
R+	R+	R+	-
F	F	F	F

4.2 Role properties

Features and transitive roles are disjoint.

```
1   (define-primitive-role s :feature t)
2   (define-primitive-role r :transitive t)
3   (related i j r)
4   (related j k r)
5   (related i j s)
6 ? (role-disjoint? r s)
```

The properties of a role induced by its parent role are depicted in Table 4.2. In the first line, if role r is declared as simple role (R) and it has a transitive role as parent, then r will not inherit transitive property from its parent.

```
1   (define-primitive-role r :parent f)
2   (define-primitive-role f :transitive t)
3 ? (transitive? r)
4 > NIL
```

If a role r is declared as a simple role and it has a feature f as a parent role, then r will be a feature itself.

```
1   (define-primitive-role r :parent f)
2   (define-primitive-role f :feature t)
3 ? (feature? f)
4    While considering r: Role contains a feature as a super role and
5    is converted into a feature.
6 > t
```

In the last column of the second line, a role declared as transitive cannot have a feature as parent:

```
1   (define-primitive-role r :transitive t :parent f)
2   (define-primitive-role f :feature t)
```

```
? (check-tbox-coherence )
      While considering r: Role contains a feature as a super role and
      is converted into a feature.
      While considering r: Role contains a feature as a super role but
      is declared to be transitive - Ignoring transitivity declaration
> (NIL NIL)
```

A feature with a transitive parent-role is not allowed:

```
  (define-primitive-role s :transitive t)
  (define-primitive-role r :feature t :parent s)
? (check-tbox-coherence )
      While considering r: Role contains a feature as a super role and
      is converted into a feature.
      While considering r: Role contains a feature as a super role but
      is declared to be transitive - Ignoring transitivity declaration
> (NIL NIL)
? (transitive? r)
> NIL
? (transitive s)
> T
```

Because `r` is a feature, the transitive property is not inherited by `r`, even if its parent `s` is transitive.

Transitive roles cannot be used in number restrictions:

```
  (define-primitive-role has-descendands :transitive t)
  (define-concept A (at-least 2 has-descendands Person))
? (check-tbox-coherence)
     Transitive role has-descendands cannot be used in number restriction
     (at-least 2 has-descendands Person)
     ignoring number restriction for role has-descendands in
     (at-least 2 has-descendands Person)
> (NIL NIL)
? (describe-concept A)
> (A
     :told-definition (at-least 2 has-descendands Person)
     :synonyms        :to-be-computed
     :parents         :to-be-computed
     :children        :to-be-computed)
? (classify-tbox)
     Concept (A) is equivalent to concept (*top* top) in TBox default.
> :OKAY
? (describe-concept A)
> (A
     :told-primitive-definition nil
     :synonyms (*top* top A)
     :parents nil
     :children ((Person)))
```

The command `(check-tbox-coherence)` is able to identify the modelling error. The macro `(describe-concept)` provides details about a concept at a specific instance of time. Note that some elements of the description are not computed yet. The function `(classify-tbox)` should be executed before querying the knowledge base. During classification, the ontology is kept consistent by removing the number restriction in the definition of concept `A`. In the current example, the definition remains `NIL`. With no constraints, the concept `A` becomes synonym with the top concept. Consequently, the other

concept `Person` in the TBox will be subsumed by `A`. Note that if `A` would be defined as `(or B (at-least 2 has-descendands Person)`, the entire definition of `A` is removed.

Similarly, roles with transitive subroles may not be used in number restrictions:

```
  (define-primitive-role has-relative)
  (define-primitive-role has-brother :transitive t :parent has-relative)
  (define-concept A (at-most 2 has-brother Person))
? (classify-tbox)
      Concept (A) is equivalent to concept (*top* top) in TBox default.
> :OKAY
? (get-concept-definition A)
> NIL
```

The concept `(all r *bottom*)` is used to refer to all individuals not being `r`-connected to any other individual.

4.3 Role queries

The macro `role-subsumes?` checks if two roles are subsumed each other. To check if two roles are equivalent, the macro `role-equivalent?` is used.

```
  (implies-role s r)
  (implies-role r s)
? (role-equivalent? r s)
> t
```

To check to properties of a role, the corresponding macros can be used: `transitive?`, `feature?`, `symmetric?`, `reflexive?`. These macros return `t` if the role property is satisfied and `NIL` otherwise.

Macro `role-inverse` returns the inverse of a role, while the domain and range are obtained with `role-domain` and respectively `role-range`.

Function `(compute-all-implicit-role-fillers)` for all individuals in current ABox. For a particular instance `i` the command `(compute-implicit-role-fillers i)` is preferred.

4.4 Concrete domains

A concrete domain attribute is a particular role having the range bound to the following types: `cardinal`, `integer`, `real`, `complex`, or `string`.

```
(define-concrete-domain-attribute has-age :type integer)
```

```
  (equivalent teenager (and human (max has-age 19)))
  (equivalent youngperson (and human (max has-age 35)))
? (concept-subsumes? youngperson teenager)
> t
  (equivalent studentWithGoodGrades (and human (>= averageGrade 8.5)))
  (equivalent studentWithVeryGoodGrades (and human (>= averageGrade 9.5)))
? (concept-subsumes? studentWithGoodGrades studentWithVeryGoodGrades)
> t
```

For the reals, RacerPro supports linear equations and inequations.

```
  (in-tbox constraints)
  (implies top (= RON  (/ EURO 4.0) ))
  (equivalent cheap-rent  (<= RON 900.0))
  (equivalent not-expensive-rent (<= EURO 300.0))
  (constrained house1 price-house1 RON)
  (constrained house2 price-house2 EURO)
  (constraints
      (= price-house1 800.0)
      (= price-house2 200.0))
? (constraint-entailed? (= price-house1 price-house2))
> t
? (concept-instances not-expensive-rent)
>
```

```
(define-concrete-domain-attribute inhabitants :type cardinal)
(define-concrete-domain-attribute has-name :type string)
(instance cluj (and city (string= has-name "Cluj-Napoca")
                         (= inhabitants 300000)))
```

The macro `constraints-entailed?` checks if certain concrete domain constraints are entailed by an ABox and a TBox.

4.5 Exercises

Exercise 4.1 (Built-in roles)**.** *Check that, after initialisation, the current TBox includes only the universal role* `*top-object-role*` *and the empty role* `*bottom-object-role*`*. Solution*

Exercise 4.2 (Modelling with DL)**.** *Model the following statements with inverse roles:*

1. *Lonely people do not have friends and are not friends of anybody.*
2. *An intermediate node is a node which has a predecessor node and a successor node.*

Solution

Exercise 4.3 (Modelling with DL)**.** *Consider the knowledge base $\mathcal{K}$:*

```
(in-tbox K)
(instance john human)
(instance suzan human)
(instance helen human)
(instance natalie human)
(instance nick human)

(instance john Male)
(instance max Male)
(instance nick Male)

(instance max Dog)

(instance john Doctor)
(instance helen Doctor)

(symmetric are-friends)
```

```
18 (related suzan helen are-friends)
19 (related natalie helen are-friends)
20 (related suzan natalie are-friends)
21 (related natalie nick are-friends)
22
23 (symmetric are-married)
24 (related suzan helen are-married)
25
26 (related john max has-pet)
27 (related suzan max has-pet)
```

1. *Represent the ABox as a labeled graph.*
2. *Represent the following concepts in KRSS syntax in ALC version of description logic. Retrieve the extensions of these concepts.*
   - *Those who are married to a doctor and have a dog as a pet.*
   - *Those who are not married and all of whose friends are either female or married men.*
3. *Represent the following sentences:*
   - *Married relationships are based on friendship;*
   - *Those who do not have male friends do not have pets;*
   - *All men are either married or have a non-male friend.*

   Are they true?

Solution

Exercise 4.4. *Check in RacerPro the implications in Table 4.1. Solution*

4.6 Solutions

Solution 4.1 (Exercise 4.1)**.**

```
1 ? (full-reset)
2 ? (all-roles)
```

Solution 4.2 (Exercise 4.2)**.**

```
1 (equivalent LonelyPerson (and Person
2                          (not (some (role-inverse has-friend) *top*))
3                          (not (some has-friend *top*))))
```

```
1 (equivalent Node (and Node
2                       (some has-next Node)
3                       (some (role-inverse has-next) Node)))
```

Solution 4.3 (Exercise 4.3)**.**

```
? (concept-instances (and (some are-married Doctor) (some has-pet Dog)))
? (concept-instances (and (not (some are-married *top*))
                          (all are-friends (or (not Male)
                                               (and Male
                                                (some are-married top))
                                           )
                           ))
```

```
? (role-subsumes? (some has-married) (some has-friend))
? (concept-subsumes? (not (some has-pet *top*)
                     (not (some are-friends Male)))
? (concept-subsumes? man (or (some are-married *top*)
                             (some are-friends (not Male))))
```

Chapter 5

Populating Ontologies

This chapter presents different ways to introduce assertions: concept assertions, role assertions, attribute assertions and constraints. It also discusses various assumptions when reasoning on ontologies: the open and closed words assumptions and the unique name assumption.

An ABox contains a set of assertions about the objects in the world and their interrelations. There are four kinds of assertions about individuals:

1. *concept assertions*, stating that an individual `i` is an instance of a concept `C`: `(instance i C)`
2. *role assertions*, stating that an individual `i` is a related by individual `j` via role `r`, with `(related i j r)`
3. *attribute assertions*, stating that an object `o` fills the role `r` of the individual `i`: `(constrained i o r)`
4. *constraints*, stating relationships between objects of a concrete domain: for instance `(constraints (= o 10))`, where `o` should be an object.

5.1 Concept assertions

The individuals asserted in an ABox are interpreted according to the definitions given in the TBox with which the ABox is associated with. The list of associated ABoxes for the current TBox is obtained with `(associated-aboxes)` and the active ABox with `(current-abox)`. To move to an existing ABox the function `(set-current-abox abox)` is used. To create a new ABox, the macro `(in-abox)` is used. The macro gets also an optional parameter to specify the name of the TBox to be associated with. Some of the functions used for concept assertions are exemplified below:

```
  (in-abox my-family
  (instance marcel Person)
  (instance ioana (and Girl Teenager))
? (all-individuals)
> (ioana marcel)
? (forget-concept-assertion racer::my-family marcel Person)
> :OKAY
? (all-individuals)
> (ioana)
? (instantiators ioana)
```

```
> ((Teenager) (*top* top) (Girl))
```

5.2 Role assertions

Through the role assertion `(related i j r)`, the individual `i` (the predecessor) is related with `j` (the filler) via the role `r`.

```
(related maria ioana has-sister)
(related maria marcel (inv has-sister))
```

The macro `(individuals-related? i j r)` returns `true` if `i` is related via role `r` with `j`.

```
  (implies-role has-friend knows)
  (related marcel ioana has-friend)
? (individuals-related? marcel ioana knows)
> t
```

Attribute assertions. The macro `(attribute-filler` sets the value for an attribute with respect to an individual:

```
 (define-concrete-domain-attribute has-age :type integer)
 (attribute-filler ioana 19 has-age)
```

Constraints. The macro `constraints` gets a sequence of concrete domain predicate assertions:

```
  (constraints (= adi-age 30) (= marcel-age 25) (> adi-age marcel-age))
? (all-constraints)
> ((> adi-age marcel-age) (= marcel-age 25) (= adi-age 30))
```

The macro `constrained` asserts that an individual is related with a concrete domain via an attribute. The three parameters are: an individual name, concrete domain name and the attribute:

```
(constrained ioana ioana-age has-age)
```

5.3 Open and Closed Word Assumptions

In the Open World Assumption (OWA) what cannot be proven to be true is not believed to be false. When RacerPro answers `NIL` it does not mean NO but just "cannot be proven given the information in RacerPro" [33].

```
  (instance i C)
  (define-individual j)
? (concepts-instances *top*)
> (i j)
? (concepts-instances C)
> (i)
? (concepts-instances (not C))
> NIL
```

```
? (concept-equivalent *top* (or C (not C)))
> t
```

In the example above, `j` could not be proven to be of type `(not C)`, even if `j` is not member of `C` and the union between `C` and `(not C)` is equivalent to the universal concept `*top*`.

```
  (instance andrei male)
  (related marcel andrei has-child)
? (concept-instances (some has-child male))
> (marcel)
? (concept-instances (all has-child male))
> NIL
```

Although the ABox contains only assertions about one male child, it is unknown whether additional information about a female child might be added later.

Unique Name Assumption (UNA). Just to specify the existence of an individual `i` in the domain, the macro `(define-individual i)` is used. To assert that an individual is distinct from all other individuals the macro `(define-distinct-individual i)` is enacted. While `(same-as i j)` states that `i` and `j` refer to the same individual, the opposite macro is `(different-from i j)`.

The macro `(abox-una-consistent?)` checks if the ABox does not contain a contradiction if the unique name assumption is imposed.

```
  (define-distinct-individual i)
  (same-as i j)
? (abox-consistent)
> t
? (abox-una-consistent?)
> NIL
```

By default, RacerPro does not assume the unique name assumption. This can be changed with:

```
(set-unique-name-assumption t).
```

To state that to individuals are the same, the macro `same-individual` is often used. For instance, to model the sentence *Venus is often called "the morning star" and "the evening star"*, one might define:

```
(same-individual theMorningStar theEveningStar)
```

Nominals. A concept can be defined by simply enumerating its instances. The operator `one-of` takes individuals and constructs a concept whose extension contains only the given individuals.

```
  (define-concept Beatle (one-of john paul george ringo))
? (all-individuals)
> (ringo george paul john)
? (concept-instances Beatle)
> (ringo george paul john)
```

ABox realization means to compute for all individuals in the ABox their most-specific concept names. The RacerPro function `(realize-abox)` also checks the consistency of the current ABox. The function should be executed before queries can be posed.

5.4 Exercises

Exercise 5.1 (Modeling with Nominals)**.** *Express, in term of subsumptions between concepts with, the following statements, using Nominals:*

- *There are exactly 7 days;*
- *Either John or Mary is a spy but not both;*
- *Alice loves either Bob or Calvin;*
- *Everything is created by God;*
- *Everybody agree in driving on the left or on the right;*

Solution

Exercise 5.2. *Consider the ABox A:*

```
(related john susan friend)
(related john andrea friend)
(related susan andrea loves)
(related andrea bill loves)
(instance susan Female)
(instance bill (not Female)
```

1. *Draw an arbitrary model of A.*
2. *Represent the following query: "Does John have a female friend who is in love with a male (not female) person?"*

Solution

Exercise 5.3 (Related individuals)**.** *Consider the ABox:*

```
(instance d A)          (related d e r)          (related d e s)
(instance e A)          (related d e r)          (related d g s)
(instance f A)          (related d e r)          (related g d s)
(instance f B)
```

1. *Draw a graphical representation of the ABox*
2. *List all individuals that are instances of the concept* `C` *defines as:*
   (a) *(or A B)*
   (b) *(some s (not A))*
   (c) *(all s A)*
   (d) *(some s (some s (some s (some s A))))*
   (e) *(not (some r (and (not A) (not B))))*
   (f) *(and (some s (and A (all s (not B)))) (not (all r (some r (or A (not A))))))*

Solution

Exercise 5.4. *Consider the graphical representation in Fig. 5.1*

1. *Formalise the corresponding ABox*
2. *List the instances of the following concepts:*
   (a) *(or A B)*
   (b) *(some s (not A))*
   (c) *(all s A)*
   (d) *(≥2 s)*
   (e) *(some s (some s (some s (some s A))))*

Exercise 5.5. *Using the individuals* `laura`, `audrey` *and* `donna`, *the concepts* `Person` *and* `NicePerson`, *and the role* `hasFriend`, *represent the following knowledge*

- *Audrey is a person.*
- *Laura is a nice person.*
- *Donna is a friend of Laura's.*
- *A nice person is a person all of whose friends are nice persons.*
- *Every nice person has a friend.*

1. *Is the statement "Donna has a friend" a logical consequence of the knowledge base KB? Explain your answer. If the answer is negative, give a model of the KB where the statement is false.*
2. *Is the statement "Audrey is a friend of Donna's" a logical consequence of the knowledge Base KB? Explain your answer. If the answer is negative, give a model of the KB where the statement is false.*

Exercise 5.6. *Model the following scenario and use RacerPro to solve the puzzle:*

- *There are three chairs on a stage*
- *On the left chair sits a girl*
- *On the right chair sits a boy*

Assuming that any child sits on the middle chair, does a boy sit next to a girl on the stage? Solution

Exercise 5.7 (Sudoku)**.** *Try to model a Suduko board with four squares. Solution*

5.5 Solutions

Solution 5.1 (Exercise 5.1).

```
(equivalent day (one-of monday thursday wednesday tuesday
                        friday saturday sunday))
```

```
(disjoint (one-of john) (one-of mary))
(implies (one-of johnormary) (one-of john, mary)))
(implies (one-of johnormary) Spy)
```

```
(implies *top* (some (inv creates) (one-of god)))
```

Solution 5.2 (Exercise 5.2).

```
(concept-instance? john (some friend (and Female
                                          (some loves (not Female)))))
```

Solution 5.3 (Exercise 5.3).

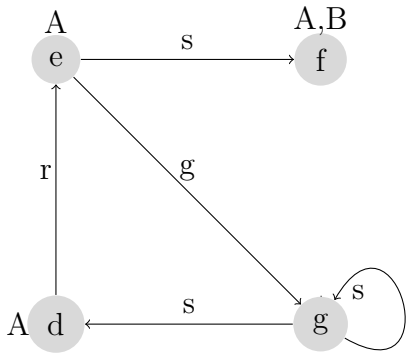

Figure 5.1: Arbitrary model for exercise 5.3.

Solution 5.4 (Exercise 5.6).

```
  (full-reset)
  (equivalent Child (or Boy Girl))
  (disjoint Boy Girl)
  (instance a Girl)
  (instance c Boy)
  (instance b Child)
  (related a b nextto)
  (related b a nextto)
  (related b c nextto)
  (related c b nextto)
  (instance s Stage)
  (related a s onstage)
  (related b s onstage)
  (related c s onstage)
? (concept-instances (and Stage (some (inv onstage)
                            (and Boy (some nextto Girl)))))
> (S)
```

It means that the boy named `s` *stands next to a girl.*

Solution 5.5 (Exercise 5.7)**.**

```
(set-unique-name assumption t)
(equivalent Field (or N1 N2 N3 N4))
(disjoint N1 N2 N3 N4)
(equivalent Group (and (exactly 1 contains N1) (exactly 1 contains N2)
                       (exactly 1 contains N3) (exactly 1 contains N4)
                       (exactly 4 contains Field)))
(instance f11 N1)
(instance f12 N2)
(instance f13 Field)
(instance f14 N4)
(instance g1 Group)
(related g1 f11 contains)
(related g1 f12 contains)
(related g1 f13 contains)
(related g1 f14 contains)
```

Chapter 6

Rules on Top of Ontology

This chapter introduces the technical instrumentation provided by RacerPro when dealing with rules. Rules are required to augment the expressivity limitation of description logic. After anyalysing the rules lifecycle, the event recognition capability is also presented.

6.1 Defining and firing rules

Rules are required because DL is not expressive enough. The meaning of such a rule is "if an individual is proved to be an instance of C, then derive that it is also an instance of D." Such rules are often called trigger rules.

Rules have a body and a head:

$$Person(?x) \vee hasParent(?x\ ?y) \vee hasBrother(?y\ ?z) \rightarrow hasUncle(?x\ ?z)$$

Defining rules in RacerPro is exemplified in the following listing:

```
1   (instance petra Person)
2   (instance alin Person)
3   (instance marcel Person)
4   (related petra alin has-parent)
5   (related alin marcel has-brother)
6   (define-primitive-role has-uncle)
7   (define-rule (?x ?z has-uncle) (and (?x Person)
8                                       (?x ?y has-parent)
9                                       (?y ?z has-brother)))
10  ? (all-rules)
11  > (:rule-1)
12  ? (related-individuals has-uncle)
13  > NIL
14  ? (run-all-rules)
15  > ((((related petra marcel has-uncle))))
16  ? (related-individuals has-uncle)
17  > ((petra marcel))
```

Because there are two different variables in the consequent, this rule cannot be translated (represented) in description logics only. The uncle concept can be represented in DL as:

```
1   (equivalent Uncle (and Man (some has-brother (some has-child *top*))))
```

Table 6.1: Rules expressible in DL.

DL statements	Rules
$A \sqsubseteq B$	$A(x) \rightarrow B(x)$
$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
$A \sqsubseteq B \sqcap C$	$A(x) \rightarrow B(x)$ *and* $A(x) \rightarrow C(x)$
$A \sqcup B \sqsubseteq C$	$A(x) \rightarrow C(x)$ *and* $B(x) \rightarrow C(x)$
$A \sqsubseteq \neg B \sqcup C$	$A(x) \wedge B(x) \rightarrow C(x)$
$A \sqsubseteq \forall R.B$	$A(x) \wedge R(x,y) \rightarrow B(y)$
$\exists R.A \sqsubseteq B$	$R(x,y) \wedge B(y) \rightarrow A(x)$
$\forall R.A \sqsubseteq A$	$(R(x,y) \rightarrow C(y)) \rightarrow A(x)$

or as a rule:

```
(define-rule (?x Uncle) (and (?x Man)
                             (?x ?y has-brother)
                             (?y ?z has-child)))
```

The translation is possible because in the consequent there is one variable. Table 6.1 lists DL statements that can be represented also as rules.

The macro `firerule` prepares a new rule by defining the antecedents, the consequent, and the premise of the rule and then fires the rule. Optionally, the ABox in which the rule is fired can be specified. Note that `forget` statements can appear in the consequence of a rule.

```
  (instance anca mother)
  (instance catrinel woman)
? (firerule (and (?x woman) (neg (?x mother))) ((instance ?x mother)))
> (((instance catrinel mother)))
? (concept-instances mother)
> (catrinel anca)
```

A common usage of rules, known as *rolification* is to assert through the conclusion of the rule a relation between two individuals.

$$Wale(x) \vee Penguin(y) \rightarrow biggerThan(x,y)$$

$$Woman(x) \vee marriedTo(x,y) \vee Man(y) \rightarrow hasHusband(x,y)$$

In RacerPro, these two rules are represented with:

```
(define-rule (?x ?y biggerThan)(and (?x Wale)(?y Penguin)))
(define-rule (?x ?y hasHusband)(and (?x Woman)(?y Man)(?x ?y marriedTo)))
```

6.2 Rule life cycle

All rules, either ready to run, currently running or which have already been processed are listed by the command `(all-rules)`. A specific rule can be deleted with `(delete-rule rule-id)` A rule can be in the following states: ready, running, waiting (sleeping) or terminated. The current status of a rule is retrieved with `(describe-current-status rule-id)`, while the status of all rules is obtained with the command `(describe-all-rules)`.

To check if a specific rule is ready the function `(rule-ready-p rule-id)` is used. If an answering thread exists for a rule, the command `(rule-active-p rule-id)` returns true. Some of the active queries stop their execution to wait for a next tuple. This status is checked with `(rule-waiting-p rule-id)`. If all tuples of a rule have been computed or a timeout occurred, a rule is considered terminated or inactive, verified with `(rule-processed-p rule-id)` or `(rule-inactive-p rule-id)`. Functions for retrieving all the rules with a particular status exist: `(ready-rules)`, `(active-rules)`, `(running-rules)`, `(waiting-rules)`, respectively `(terminated-rules)`.

The preconditions of a rule have to be satisfied in "all worlds". If the all antecedents of a rule are satisfied the rule is applicable, verified with `(rule-applicable-p rule-id)`. Applicable rules are either ready rule or a processed rule. Active rules are not applicable. This can be checked with the commands `(applicable-rules)` and `(unapplicable-rules)`.

Execution control. The function `(swrl-forward-chaining)` fires SWRL (Semantic Web Rule Language) rules until no new information is added. A given rule is fired with `(execute-rule rule-id)`. The function `(execute-all-rules)` maps `(execute-rule)` over the list ready rules. A rule can be forced to be fired more than once with the command `(reexecute-rule rule-id)`.

The user can implement its own rule application strategy or can call the function `(add-rule-consequences-automatically)`. This can be disabled with the RacerPro command `(dont-add-rule-consequences-automatically)`. The set of rule consequences is added to the ABox only after the rule terminates. Note that rules are not executed if the ABox becomes inconsistent.

6.3 Event rules

The main assumption when reasoning on events is that assertions only hold for a period of time.

Consider the vehicle lane changing scenario below:

```
1  (instance c1 Vehicle)
2
3  (define-event-assertion ((hasLocation c1 l1) 0 1))
4  (define-event-assertion ((hasLocation c1 l1) 1 2))
5  (define-event-assertion ((hasLocation c1 l2) 2 3))
6
7  (instance l1 (and (= hasLat 49) (= hasLong 16)))
8  (instance l2 (and (= hasLat 51) (= hasLong 16)))
9
10 (equivalent Lane1 (and (< hasLat 50) (> hasLat 48) (= hasLong 16)))
11 (equivalent Lane2 (and (< hasLat 52) (> hasLat 50) (= hasLong 16)))
```

The vehicle `c1` has location `l1` in the time interval [0,2] (lines 3 and 4). Between time instance 2 and 3, the vehicle `c1` appears on lane `l2` of the highway (listing line 5). The following rule can be used to recognise the basic lane changing event:

```
1  (define-event-rule ((laneChange ?v ?l1 ?l2) ?t0 ?t2)
2     ((?v vehicle) ?t0 ?tn)
3     ((?l1 Lane1) ?t0 ?tn)
4     ((?l2 Lane2) ?t0 ?tn)
5     ((hasLocation ?v ?l1 ) ?t0 ?t1)
6     ((hasLocation ?v ?l2 ) ?t1 ?t2))
```

The rule has five premises and one consequence - the `laneChange` event.

When debugging a rule, particular attention should be paid that premises are satisfied. For instance:

```
> (concept-instances Vehicle)
> (C1)
? (concept-instances Lane1)
> (L1)
? (concept-instances Lane2)
> (L2)
```

If the rule fires is checked with:

```
? (timenet-retrieve ((laneChange ?v1 ?obj1 ?obj2) ?t0 ?t2))
> (((?V1 C1) (?OBJ1 L1) (?OBJ2 L2) (?T0 (1 1)) (?T2 (3 3))))
```

The answer confirms that vehicle `c1` has changed the line `l1` with `l2` between time instance 1 and 3.

6.4 Exercises

Exercise 6.1 (Translating DL into rules)**.** *Represent the following DL expression as rules:*

- *(implies A B)*
- *(role-implies r s)*
- *(implies (and A (some r (some s B))) C)*
- *(implies A (all r B))*
- *(implies A (not B))*
- *(implies A (or (not B) C))*
- *(implies *top* (at-least 1 r *top*))*

Solution

Exercise 6.2 (Rolification)**.** *Represent the following rule in RacerPro:*

$$worksAt(x,y) \wedge University(y) \wedge supervises(x,z) \wedge PhDStudent(z) \rightarrow professorOf(x,z)$$

Exercise 6.3 (Rules and Lisp)**.** *To each person in the current ABox without a known age, assign an age between 18 and 100. Solution*

Exercise 6.4 (Rolification)**.** *Use rules to define an* ***identity*** *role, that is a primitive role that relates an individual with itself.*

6.5 Solutions

Solution 6.1 (Exercise 6.1)**.**

```
(define-rule (?x B) (?x A))
(define-rule (?x ?y s) (?x ?y r))
(define-rule (?x C) (and (?x A) (?x ?y r) (?y ?z s) (?z B)))
(define-rule (?y B) (and (?x A) (?x ?y r)))
(define-rule *bottom* (and (?x A) (?x B)))
(define-rule (?x C) (and (?x A) (?x B)))
(define-rule (?y ?z same-as) (and (?x ?y r) (?x ?z r))) -- to be checked
```

Solution 6.2 (Exercise 6.3)**.**

```
  (define-datatype-property age :range integer)
  (instance i Person)
  (instance j Person)
  (instance k Person)
  (add-datatype-role-filler racer::default k 20 age)
? (firerule (and (?x Person) (neg (?x (an age))))
            ((lambda (defer (add-datatype-role-filler (current-abox)
                        ?x (+ 18 (random 82)) 'age)))))
> (((PROGN
           ((:LAMBDA
             NIL
             (ADD-DATATYPE-ROLE-FILLER 'DEFAULT 'j '45 'age))
            NIL)))
        ((PROGN
           ((:LAMBDA
             NIL
             (ADD-DATATYPE-ROLE-FILLER 'DEFAULT 'i '49 'age))
            NIL))))
? (individual-told-datatype-fillers j age)
> (45)
```

Chapter 7

Ontology Design Patterns

This chapter aims to present the student some ontology design solutions used for recurrent use cases, called ontology design patterns. Examples are provided for structural, correspondence, content, and presentation ontology design patterns.

An ontology design pattern (ODP) is a modelling solution to solve a recurrent ontology design problem [18]. ODPs are small ontologies that provide design solutions for recurrent use cases. ODPs facilitates knowledge reuse and ontology modularisation. Ideally, an ontology is composition of various ODPs. The use of ODPs is a sign of good ontology quality [16].

Following [18], we distinguish six types of design patterns: *structural*, *correspondence*, *content*, *reasoning*, *presentation*, and *lexico-syntactic*. Each category includes various ODPs, forming a taxonomy of ODPs. Fig. 7.1 formalises this taxonomy in the KRSS syntax.

```
1 (equivalent ODP (or StructuralODP CorrespondenceODP ContentODP
2                     ReasoningODP PresentationODP LexicoSyntacticODP))
3 (equivalent StructuralODP     (or LogicalODP ArchitecturalODP))
4 (equivalent CorrespondenceODP (or ReeingineeringODP MappingODP))
5 (equivalent PresentationODP   (or NamingODP AnnotationODP))
```

Figure 7.1: The top level taxonomy of ODPs.

For a compplete list of ODP, the reader is encouraged to browse ODPs repositories: http://www.gong.manchester.ac.uk/odp/html/ and ontologydesignpatterns.org/.

7.1 Structural ODPs

Structural ODPs include Logical ODP and Architectural ODP (see Fig 7.1).

Logical ODPs

Logical ODPs solve design problems where the primitive of the ontology language used do not directly support specific logical constructs [18]. As an example, consider that the ontology engineer needs to represent a relation between more than two elements. Recall that DL provides as primitive elements concepts and binary roles. A logic ODPs is helpful to model the semantics of the n-ary relation required in this example. Thus, logical ODPs are content-independent structures addressing a problem of expressivity. These patterns

are expressed only by means of a logical vocabulary of description logic. Two logical ODPs are detailed: *N-ary relation* and *Partition ODP*, as introduced in [18].

N-ary relation. The aim is to model n-ary relations in an ontology, given that DL was designed to express binary relations.

```
(implies N-aryRelationship (and (some property-1 Attribute-1)
                                (some property-2 Attribute-2)
                                (some property-n Attribute-n)))
```

Consider the relation *Employee(name, age, address, position)*. We introduce concept `Employee` to represent the relation. For each argument of the relation, we introduce either a role or a concrete domain attribute. Here we introduce the attribute `has-name` with the fillers of type `String`, the attribute `has-age` with fillers of type `Integer`, and two roles: `has-address`, respectively `has-position`. For each introduced role, we define a class representing the range of the role. Thus, we introduce the concepts `Location` for the range of role `has-address` and the concept `JobType` for the role `has-position`.

```
(define-concept Employee)
(define-concrete-domain-attribute has-name :domain Employee :type string)
(define-concrete-domain-attribute has-age :domain Employee :type integer)
(define-primitive-role has-address :domain Employee :range Location)
(define-primitive-role has-position :domain Employee :range JobType)

(instance e-170 Employee)
(attribute-filler e-170 "Joe" has-name)
(attribute-filler e-170 22 has-age)
(related e-170 cluj has-address)
(related e-170 programmer has-position)
```

Partition pattern. It is natural to consider whether some set of subconcepts fully covers a concept. For example, we might want to say that:

```
(equivalent Parent (or Mother Father))
(disjoint Mother Father)
```

In an ontology, a partition is a concept which is divided into several disjoint subconcepts. The general form of the partition pattern in KRSS is:

```
(define-concept P (or C0 C1))
(disjoint (C1 C2))
```

Selector. A selector is a modifier that can be used to select between symmetrical entities (right and left hand, anterior-posterior). The functional role `has-selector` is used to add a selector to the classes of the domain hierarchy (e.g. hand can be left or right).

```
(equivalent Selector (or Selector1 Selector2 Selector3))
(disjoint Selector1 Selector2 Selector3)
(functional has-selector)
(implies AffectsEntitySelectorValue3
     (some affects (and Entity (some has-selector SelectorValue3))))
(implies Entity (some has-selector Selector))
```

```
(equivalent Laterality (or Left Right))
(disjoint Left Right)
(functional has-laterality)
(implies RightHandBurn
    (some affects (and (some Hand has-laterality Right))))
(implies Hand (some has-laterality Laterality))
```

Architectural ODPs

Architectural ODPs constraint how the ontology should look like. These patterns are a composition of Logical ODPs used to affect to shape of the ontology. Three instances of architectural ODPS are: *taxonomy*, *lightweight* or *modular*.

Modular ontology. Based on this pattern, an ontology consists of several modules connected by the *import* operator. The function `(kb-ontologies KB)` retrieves all ontologies that were imported into the specified OWL knowledge base `KB`. The macro `(in-knowledge-base T-box)` is a shortcut for `(in-tbox T-box)` and `(in-abox A-box T-box)`. The `init` argument of the `owl-read-file` command specifies whether the knowledge base is initialised (`t`) or extended (`nil`).

```
? (in-knowledge-base University)
> (University University)
? (instance i C)
> :OKAY
? (save-kb "/home/adrian/university.owl" :syntax :owl
                                         :if-exists :supersede :uri "")
> :OKAY
? (in-knowledge-base Institution)
> (Institution Institution)
? (instance j D)
> :OKAY
? (save-kb "/home/adrian/institution.owl" :syntax :owl
                                          :if-exists :supersede :uri "")
> :OKAY
? (owl-read-file "/home/adrian/university.owl" :init nil
                                               :kb-name 'Institution)
? (current-tbox)
> ./university.owl
? (all-tboxes)
> (OWLAPI-KB Institution default University)
? (all-individuals)
> (./university.owl#i j)
? (all-atomic-concepts)
> (top bottom /home/adrian/university.owl#C D)
```

7.2 Correspondence ODPs

Two categories of patterns are classified as correspondence ODP: *reingineering* ODPs and *mapping* ODPs. Reengineering ODPs provide support to transform a conceptual model into an ontology. Mapping ODPs provide solutions to create associations between elements of two ontologies.

Reengineering ODPs

Reengineering ODPs support the transformation of non-ontological resources (i.e., thesauri, database schemas) into ontologies. These patterns apply transformation rules to a source model to create a new ontology from these sources. The source model can be: text corpora, lexicographic resources (dictionaries, wordnets, terminologies), various knowledge organization systems (thesauri, classification schemes), folksonomies (tag sets, directories, topic trees, subject indexes) frames, semantic networks, microformats, infoboxes, HTML templates or DB schemas and records. The resulted ontology covers the terminology/metadata/textual corpora encapsulated in the input resources. Thereby, the resulted ontology is a coverage-oriented ontology.

Mapping ODPs

Mapping ODPs enact three basic semantic relation between the elements of two ontologies: *equivalence*, *containment* and *overlap*, supplemented with their negative counterparts.

7.3 Content ODPs

Content ODPS are instances of LPs or compositions of LPs. CODPs are content dependent because they rely on vocabulary from a specific domain of interest. They are small ontologies that mediate between use cases and design solutions. Ideally, an ontology should be a composition of CODPs.

Examples are: *PartOf*, *Participation*, *Plan*, *Legal Norm*, *Legal Fact*, *Sales Order*, *Research Topic*, *Legal Contract*, *TimeInterval*, etc. For a more comprehensive list, the reader is refereed to http://ontologydesignpatterns.org/wiki/Category:ProposedContentODP.

PartOf. This ODP aims to represent entities and their parts. The pattern exploits reasoning on transitive role `has-part`. In the example below, the question is if the brain is part of the organism.

```
1   (define-primitive-role has-part :transitive t
2                                   :inverse is-part-of
3                                   :domain Entity)
4   (implies Organism (some has-part NervousSystem))
5   (implies Brain (some is-part-of NervousSystem))
6   (implies Brain Entity)
7 ? (transitive? is-part-of)
8 > T
9 ? (concept-subsumes? (some is-part-of NervousSystem) Brain)
10> T
```

The role `directPartOf` is a subrole of `partOf` but it does not inherit transitivity. Note that in RacerPro the transitive property is not inherited:

```
1   (define-primitive-role partOf :transitive t)
2   (define-primitive-role directPartOf :parent partOf)
3 ? (transitive? directPartOf)
4 > nil
```

Table 7.1: Naming ODPs.

Naming ODP	Convention	Example
Namespace	the name of the organisation that publishes the ontology	http://cs-gw.utcluj.ro/∼isg
Class names	no plurals useful to include the name of the parent concepts as suffix	*Students* is considered bad practice `(implies LeftHand Hand)`
Roles	usually start with a lower case letter	`partOf`, `hasChild`

AgentRole. The pattern permits assertions on roles played by agents without involving the agents that play that roles, and vice versa. As an example consider the sentence *"George Enescu was a Romanian composer. He was also conductor and teacher."* and the corresponding KRSS representation below:

```
1  (in-tbox agentrole_ex1)
2  (disjoint Role Agent)
3  (implies top (all hasRole Role))
4  (implies top (all isRoleOf Object))
5  (implies Role Concept)
6  (implies Agent Object)
7
8  (in-abox agentrole_ex1)
9  (instance composer Role)
10 (instance conductor Role)
11 (instance teacher Role)
12 (instance georgeEnescu Agent)
13 (related georgeEnescu composer hasRole)
14 (related georgeEnescu conductor  hasRole)
15 (related georgeEnescu teacher hasRole)
```

7.4 Presentation ODPs

Presentation ODPs are templates for annotation and documentation to support ontology readability.

Naming. The pattern refers to conventions about how to create names ontology elements aiming to increase ontology readability by the human agent (see Table 7.1).

7.5 Working with ODPs

The following operations are used to enact an ODP [18, 47]:

- *Import*: incudes an ODP in the ontology under development, without modifying its elements. The importing ontology benefits from the set of inferences provided by the ODP.
- *Specialisation*: consists of creating sub-concepts or sub-roles from the elements of an ODP. An odp_1 is a specialisation of odp_2 if at least one ontology element from odp_1 is subsumed by by an ontology element from odp_2.

- *Generalisation*: An odp_1 is a generalisation of odp_2 if at least one ontology element from odp_1 subsumes an ontology element from odp_2.
- *Composition*: consists of associating concepts (roles) of an ODP with concepts (roles) of other ODPs, by means of some DL axioms.
- *Expansion*: consists of adding new concepts, roles or axioms to cover the requirements that are not addressed by the reused ODP.

7.6 Exercises

Exercise 7.1 (Types of ODPs)**.** *This exercise helps you to develop a conceptual map of ODPs:*

1. *Extend the taxonomy introduced in Fig. 7.1 with various ODPs that are documented in the Semantic Web literature.*
2. *Propose some roles that can be associated with the concept ODP in order to clarify yourself the aim, main usage, etc. of each design pattern.*
3. *Formalise these roles in the KRSS syntax. Include these roles in your extended taxonomy of ODPs. Now, you have a meta-ontology about ODPs.*
4. *Populate your meta-ontology with assertions about two ODPs.*

Solution

Exercise 7.2 (Types of ODPs)**.**

1. *Build your own collection of ODPs in KRSS syntax.*
2. *Build a small ontology which includes at least four ODPs from your collection.*

Solution

Exercise 7.3 (N-ary relation ODP)**.** *Represent the following n-ary relationships in RacerPro:*

- *Exam(Topic, Grade, Student, Date)*
- *Hotel(Name, Cathegory, Location)*
- *Movie(Name, Director, Producer, Length, MainActors(Family Name, Given Name))*

Exercise 7.4 (Selector ODP)**.** *Enact the Selector ODP to select one day for a meeting among 3 possible days: Monday, Tuesday, Wednesday.*

Exercise 7.5 (Working with ODPs)**.** *Specializes the co-participation ODP for representing:*

1. *an academic lecture*
2. *a football match*

Exercise 7.6 (Working with ODPs). *Represent, in KRSS syntax, knowledge about the courses offered in a your university. In particular, the following aspects should be taken into account:*

- *Courses can be regular lecture courses, seminars, or laboratory work;*
- *Each course has a syllaby from a scientific subject;*
- *Regular lecture courses have lectures with PhD. degree;*
- *Each course has a required grading work that can either be an exam, one or more projects, or both;*
- *Each course is associated with a set of faculty members;*

Using ODP, develop the TBox and ABox that represent the courses offered by your department with the corresponding faculty members. Search in your ontology for:

- *All faculty members with PhD.*
- *Is there any course lectured by a faculty member without PhD. degree?*
- *All courses from a particular scientific domain like Artificial Intelligence or Networks.*

Exercise 7.7 (Linguistic ODP). *Use ODPs to represent in KRSS syntax, the following information, adapted from Wikipedia:*

> *"The Solar System comprises the Sun and its planetary system of eight planets, as well as a number of dwarf planets, satellites, and other smaller objects that orbit the Sun. The four smaller inner planets, Mercury, Venus, Earth and Mars, also called the terrestrial planets, are primarily composed of rock and metal. The four outer planets, called the gas giants, are substantially more massive than the terrestrials. The two largest, Jupiter and Saturn, are composed mainly of hydrogen and helium; the two outermost planets, Uranus and Neptune, are composed largely of substances with relatively high melting points (compared with hydrogen and helium), called ices, such as water, ammonia and methane, and are often referred to separately as "ice giants". other small-body populations including comets, centaurs and interplanetary dust freely travel between regions. Six of the planets, at least three of the dwarf planets, and many of the smaller bodies are orbited by natural satellites, usually termed "moons" after Earth's Moon. Each of the outer planets is encircled by planetary rings of dust and other small objects".*

Exercise 7.8 (Linguistic ODP). *Use ODPs to represent in KRSS syntax, the following information, adapted from wikipedia:*

> *"Artificial intelligence (AI) is technology and a branch of computer science that studies and develops intelligent machines and software. The central problems (or goals) of AI research include reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects. General intelligence (or "strong AI") is still among the field's long term goals. Currently popular approaches include statistical methods, computational intelligence and traditional symbolic AI".*

Exercise 7.9 (Working with ODPs)**.** *Consider the following sentence:*

> *"Marcel Iures is Berenger in the play of "Rhinoceros" by Eugen Ionesco, that is given at the theater "National Theater of Cluj-Napoca" during January and February 2015."*

Detect the modeling issues in the sentence and match them to the corresponding ODPs.

7.7 Solutions

Solution 7.1 (Exercise 7.1)**.**

1. *Following [18] a possible extension would be:*

```
(equivalent LogicalODP (or LogicalMacroODP Transformation_ODP))
(implies All_and_AnyODP LogicalMacro_ODP)
(implies N-ary_relationODP Transformation_ODP)
(equivalent ArchitecturalODP (or InternalODP External_ODP))
(implies ClassificationODP ReasoningODP)
(implies SubsumptionODP Reasoning ODP)
(implies MaterializationODP Reasoning ODP)
(implies De-anonymizingODP Reasoning_ODP)
(equivalent ReengineeringODP (or Schema_RengineeringODP
                                  RefactoringODP))
(implies (or EquivalenceODP ContainmentODP OverlapODP) MappingODP)
(implies (or Not_EquivalenceODP NotContainmentODP NotOverlapODP)
                                                      Mapping_ODP)
```

2. *Possible roles are:* `has-name`*, for naming the pattern,* `has-intent`*, for describing the generic use case of the pattern,* `has-competency-question`*, for listing the knowledge that should be addressed by the pattern,* `has-scenario`*, for specifying requirements in natural language,* `has-elements`*, for enumerating the concepts and roles included in the pattern,* `has-consequence`*, for describing the benefits of the pattern,* `known-usage`*, for listing the known ontologies enacting the pattern,* `related-pattern`*, for indicating other patterns that are a specialisation, generalisation, composition, or component of the current pattern,* `has-URI`*, for referring to the implementation of the pattern (the OWL file).*

Solution 7.2 (Exercise 7.2)**.**

1. *An initial library of ODPs is already available http://www.ontologydesignpatterns.org.*

```
(OWLAPI-readOntology "/path/pattern.owl" :maintain-owlapi-axioms t)
(save-kb "/path/patern.racer" :syntax :krss)
```

Solution 7.3 (Exercise 7.9)**.** *The solution is inspired from [47]. Firstly, from the paragraph "Marcel Iures is Berenger in the play..." the ontology engineer has to model that "a person play a character". This use case is added to the Problem Space. A the end of the analysis, for each item in the Problem Space is tried to be matched against an ODP. The set of available ODPs form the Solution Space. Until now, the engineer should identify an ODP which is able to represent objects and the roles they play. Specifically, the engineer*

should browse patterns containing elements such as: concepts `Role` *and* `Object` *and the relationship* `hasRole` *with its inverse* `isRoleOf`*.*

Secondly, from the text "the play of Rhinoceros", the modelling issue is "the play of some drama". Here, 'drama' is an information object, while Rhinoceros is its concrete realisation. The adequate pattern should contain elements such as the role `isRealisedBy` *and the concepts* `InformationRealisation` *and* `InformationObject`*.*

Thirdly, the text "during January and February 2015" introduces in the Problem Space the modelling issue "time period". The engineer should identify in the Solution Space patterns able to represent time intervals with the corresponding start/end dates. Technically, the ODPs should constant ontological elements like: the concept `TimeInterval` *which has a single start date and end date and it represents the domain of some roles like* `hasIntervalDate`*,* `hasIntervalEndDate`*, or* `hasIntervalStartDate`*.*

At this moment, the given sentence is modelled with three ODPs:

> *"⟨Marcel Iures is Berenger⟩$_{ODP1}$* ***in*** *⟨the play of "Rhinoceros" by Eugen Ionesco, that is⟩$_{ODP2}$* ***given at*** *the "National Theatre of Cluj-Napoca"* ***during*** *⟨January and February 2015⟩$_{ODP3}$."*

The abstract model so far is:

> *An actor plays a character in a play, given at a theater during a time period.*

The question is if can we identify an ODP able to represent this abstract model? We can use an ODP designed to represent a situation and its corresponding circumstances.

Problem space	*Solution Space*
a person play a character	*Agent role*
the play of some drama	*Information realisation*
time period	*Time interval*
an actor plays a character in a play, given at a theater during a time period	*Situation*

Chapter 8

Debugging Ontology

This chapter introduces the technical instrumentation required to identify and solve errors in the ontology. First, common inconsistency errors are exemplified. Then, the satisfiability is analysed from different perspectives: concepts satisfiability, role satisfiability and TBox satisfiability. The learner is advised how to identify cycling definitions. Having an image on the errors in the current ontology, the discourse focuses on how to repair these errors. The strategy enacted in the chapter is first to obtain explanation about the error source and then to repair the definitions which contribute the most to the error.

8.1 Consistency checking

An ontology has a model if there is an interpretation which satisfies all the sentences in the ontology. If an ontology does not have a model, it is *inconsistent*.

Common inconsistency errors are:

1. an individual belongs to a concept and its complement.
2. an individual belongs to disjoint concepts.
3. an individual is an instance of unsatisfiable concepts.
4. an individual has a max (min) cardinality restriction but related to more (less) individuals.
5. violation of domain or range restrictions.

Other common mistakes are *partition errors*. A first example of partition error is when an individual belongs to more than one subclass of a defined partition. If the concept `Pet` is partitioned in `Dog` and `Cat`, an inconsistency occurs if we define `Tom` as an instance of both classes.

```
1 (implies (or Dog Cat) Pet)
2 (disjoint Dog Cat)
3 (instance tom Dog)
4 (instance tom Cat)
```

A second example is when a concept is a subclass of more than one subclass of a defined partition. If the concept `Mammal` is partitioned in `Dog` and `Cat`, an inconsistency occurs if the concept `Pet` is a subclass of both `Dog` and `Cat`.

```
(implies (or Dog Cat) Mammal)
(disjoint Dog Cat)
(implies Pet Dog)
(implies Pet Cat)
```

A third situation of partition error occurs when a concept `C` is exhaustive partitioned into several subclasses and an instance `i` of `C` does not belong to any subclasses of `C`. If the concept `Numbers` is exhaustively partitioned into `Odd` and `Even`, and `Four` is an instance of `Numbers`, but not an instance of `Even` or `Odd`.

```
(equiv Numbers (or Even Odd))
(disjoint Even Odd)
(instance four Numbers)
(instance four (not Even))
(instance four (not Odd))
```

8.2 Satisfiability

Concept satisfiability. A concept `C` is satisfiable with respect to a TBox if there exists a model in which $C^{\mathcal{I}}$ is not empty. The following sentences hold:

C is unsatisfiable $\equiv C \sqsubseteq \bot$
C and D are disjoint $\equiv C \sqcap D$ is subsumed by $\bot$
C and D are disjoint $\equiv C \sqcap D$ is unsatisfiable

The macro `(concept-satisfiable? C)` returns `T` if `C` is satisfiable, `NIL` otherwise.

```
  (equivalent parent (and person (some hasChild Person)))
  (equivalent woman (and female person))
  (equivalent mother (and female parent))
? (concept-satisfiable? (and (not woman) mother))
> NIL
```

By unfolding the definition of the concept $\neg Woman \sqcap Mother$ we obtain:

$$\begin{aligned}
\neg Woman \sqcap Mother &\equiv \neg(Female \sqcap Person) \sqcap Female \sqcap Parent \\
&\equiv (\neg Female \sqcup \neg Person) \sqcap Female \sqcap Parent \\
&\equiv (\neg Female \sqcup Female) \sqcap (\not Person) \sqcap Female) \sqcap Parent \\
&\equiv (\neg Person) \sqcap Female) \sqcap Parent \\
&\equiv (\neg Female \sqcup Female) \sqcap (\neg Person) \sqcap Female) \sqcap Person \\
&\quad \sqcap \exists hasChild.Person \\
&\equiv \bot
\end{aligned}$$

Given the TBox:

```
  (in-tbox inconsistent-concept)
  (implies A B)
  (implies (and A B) *bottom*)
  (implies B C)
? (tbox-coherent)
> Concept (A) is incoherent in TBox inconsistent-concept
> nil
```

The second axiom states that $A^{\mathcal{I}} \cap B^{\mathcal{I}} \subseteq$, hence `A` and `B` do not share instances. The first axiom says that all instances of `A` are also instances of `B` ($A^{\mathcal{I}}$). Thus, the concept `A` is unsatisfiable.

Role satisfiability. The empty role is unsatisfiable. Thereby, any role `r` subsumed by the empty role is also unsatisfiable.

```
1 ? (role-satisfiable? *bottom-object-role*)
2 > NIL
```

Checking if a role R is satisfiable is equivalent to checking if the concept $\geq 1.R$ or $\exists R.\top$ is satisfiable.

TBox satisfiability. A TBox `KB` is satisfiable if, there is a model of `KB`. The macro `(tbox-coherent)` returns true if there is an inconsistent atomic concept in the current TBox.

The function `(check-tbox-coherence &optional (tbox (current-tbox)))` returns a list of all atomic concepts in the given TBox that are not satisfiable.

The function `(tbox-cyclic-p &optional (tbox (current-tbox)))` returns true if the specified tbox contains cyclic CGIs.

RacerPro tries to optimise TBoxes by transforming GCI into primitive concept definitions. The definitions compiled by RacerPro are obtained with the macro:

```
(get-concept-definition CN &optional (TBN (current-tbox)))
```

This function `(realize-abox &optional (abox (current-abox)))` checks the consistency of the ABox and computes the most specific concepts for each individual in the ABox. The function `(abox-consistent-p &optional (abox (current-abox)))` returns `T` if the ABox is consistent and `nil` otherwise.

Cyclic definitions. Definitions are cyclic in the sense that concepts are either defined in terms of themselves or in terms of other concepts that indirectly refer to them.

```
1   (implies A B)
2   (implies B (and C (some r D)))
3   (implies C (and A (all r E)))
4 ? (tbox-cyclic?)
5 > ((C) (B) (A))
6 ? (concept-satisfiable? A)
7 > T
```

Note that a cyclic concept can be satisfiable.

Cyclic GCI can be either explicitly defined or can implicitly result from processing. For instance, in some cases domain or range restrictions have the consequence that cycles occur in the TBox.

8.3 Ontology repair

The strategy for ontology repair is first to obtain explanation about the error source and then to repair the definitions which contribute the most to the error.

Explanation. Explanation aims to discover why certain (possibly unwanted) consequences follow from the ontology. A *justification* is a minimal explanation for some ontology consequence. A justification for an unsatisfiable concept is a minimal subset of the ontology from which it follows that the concept is unsatisfiable. Consider the justification for unsatisfiable concept below:

A_1: `(implies A B)`
A_2: `(implies (and A B) *bottom*)`
A_3 `(implies B C)`

The justification $\{A_1, A_2\}$ is the minimal subset of axioms in the ontology which supports the unsatisfiability of concept `A`.

The `(check-tbox-coherence)`, `(check-abox-coherence)` and `(check-ontology)` commands are empowered with some explanation capabilities. The following explanations are retrieved when checking the coherence of an ABox:

```
1   (instance a (all r C))
2   (instance b (not C))
3   (related a b r)
4 ? (check-abox-coherence)
5 > (NIL ((a (all r C)) (b (not C)) ((b a) (inv r)) ((a b) r)))
```

The argument `explain-all` of the command `check-ontology` should be set on `t`. The general command `(retrieve-with-explanation)` is used when a boolean query is answered with `nil` [33].

Repair. Repair aims to provide methods to modify the ontology in an intelligent way, in order to eliminate the unwanted consequences. A repair for an error in an ontology $\mathcal{O}$ is the minimal subset of axioms from $\mathcal{O}$ which need to be removed in order the error not do hold anymore. Because more than one repair may exist for the same error, the repair with the lowest impact should be selected.

Resolving errors. In many cases the unsatisfiability of a concept triggers the unsatisfiability of another concept.

```
1   (implies A (not D))
2   (implies B D)
3   (implies A (and B (some r E)))
4   (equivalent C A)
5   (implies F (or A C))
6 ? (classify-tbox)
7 > Concept (A) is incoherent in TBox DEFAULT.
8   Concept (F) is incoherent in TBox DEFAULT.
9   Concept (C) is equivalent to concept (*BOTTOM* BOTTOM A F)
10  in TBox DEFAULT.
11> :OKAY
```

Observe that the function `(classify-tbox)` returns only unsatisfiable atomic concepts. For the non atomic concept F, we can use:

```
1 ? (concept-satisfiable? F)
2 > NIL
```

By repairing the unsatisfiability of `A` first, the concepts `C` and `F` may be automatically repaired. From axioms `(2)` and `(3)`, it holds that `(implies A D)` which conflicts with axiom `(1)`. By removing axioms `(1)`, all the concepts are satisfiable.

```
? (forget (:tbox default) (implies A (not D)))
> :OKAY
? (check-tbox-coherence)
> (NIL NIL)
```

Unwanted axioms. Consider the example:

```
(instance tweety Penguin)
(implies Penguin Bird)
(implies Bird FlyingAnimal)
```

The ontology entails that `tweety` is a flying animal:

```
? (concept-instances FlyingAnimal)
> (tweety)
```

The ontology engineer together with the domain expert are responsible to identify which of the inferences are desired or not desired.

Rewriting axioms. For computing fine-grained justifications, each complex axiom is rewritten into multiple simpler axioms, which together capture the original meaning of the complex axiom. For a given concept name the macro `get-concept-definition` returns the definition compiled by RacerPro by transforming GCIs into primitive concept definitions.

```
  (implies C (some r B)))
  (implies C D)
? (get-concept-definition C)
> (AND D (SOME r B))
? (implies C (all r (not B)))
> :OKAY
? (get-concept-definition C)
> (AND (ALL r (NOT B)) (AND D (SOME r B)))
?  (concept-satisfiable? C)
> NIL
```

Note that the aggregated definition of `C` bears out that `C` is not satisfiable.

8.4 Exercises

Exercise 8.1 (Inconsistency). *Is there any inconsistency in the following declarations?*

```
(define-distinct-individual yeti)
(same-as yeti snowman)
```

Solution

Exercise 8.2 (Inconsistency). *Check if concept `C` is satisfiable or cyclic, given the GCI:*

```
(implies C (some r C))
```

Solution

Exercise 8.3 (Inconsistency). *Check the consistency of the following TBox:*

```
(implies *top* (and (some r A) (some s B)))
```

Draw three possible models of the TBox.

Exercise 8.4 (Inconsistency)**.** *Check the consistency of the following TBox:*

```
(implies B D)
(implies A (not D))
(implies A (and B (some R C)))
```

Which concept is unsatisfiable? Can you weaken axiom A_3 such that the concept remains unsatisfiable? (an axiom A' is weaker than an axiom A iff $\{A\} \models A'$ and $\{A'\} \not\models A$). Solution

Exercise 8.5 (Inconsistency)**.** *Give your own examples of inconsistency, for each of the following common errors:*

1. An individual belongs to a class and its complement;
2. An individual belongs to disjoint classes;
3. An individual is an instance of unsatisfiable concepts;
4. An individual has a max (min) cardinality restriction but related to more (less) individuals;
5. Violation of domain or range restrictions.

Exercise 8.6 (Debugging)**.** *Consider the knowledge base:*

```
(implies (not (or A B)) *bottom*)
(implies A (and (not B) (some r B)))
(implies D (all r A))
(implies B (and (not A (some r A))))

(related a b r)
(related a c r)
(related a d r)
(related a d c)
(instance a (and B (all r D)))
(instance b E)
(instance c (not A))
(instance d (some s (not D)))
```

- *Check for the Tbox*
- *Check for the Abox*
- *Check for the KB*

is satisfiable.

Exercise 8.7. *Find the inconsistency in the following knowledge base:*

```
(implies Reindeer Mammal)
(implies (and Mammal Flies) Bat)
(implies Bat (all worksFor (oneof batman)))
(instance rudolf (and Reindee (some hasNode Red)))
(instance santa (all (inv worksFor) (or Flies (not Reindeer))))
(related rudolf santa workFor)
(different-from santa batman)
```

8.5 Solutions

Solution 8.1 (Exercise 8.1)**.** *Checking the consistency of the current ABox, RacerPro answers true:*

```
? (abox-consistent?)
> T
```

This is because, by default, RacerPro does not assume the unique named assumption. It means that the individual `yeti` *might be identified with other named individuals. To solve this, one might use:*

```
? (full-reset)
> :OKAY-FULL-RESET
? (set-unique-name-assumption t)
> T
? (define-distinct-individual yeti)
> :OKAY
? (same-as yeti snowman)
> :OKAY
? (abox-consistent?)
> NIL
```

Solution 8.2 (Exercise 8.2)**.**

```
? (concept-satisfiable? C)
> T
? (tbox-coherent?)
>  Concept (C) causes a cycle in TBox DEFAULT
```

Solution 8.3 (Exercise 8.4)**.**

```
? (tbox-coherent?)
> Concept (A) is incoherent in TBox DEFAULT
```

The third axiom can be replaced by the weaker version, in the modified TBox

```
  (full-reset)
  (implies B D)
  (implies A (not D))
  (implies A B)
> ? (concept-satisfiable? A)
> NIL
```

Concept `A` *remains unsatisfiable with the weaker axiom.*

Chapter 9

Ontology Evaluation

This chapter provides the conceptual instrumentation needed to evaluate an ontology. The ontology evaluation metrics detailed in this chapter deals with both structural evaluation an semantic evaluation. The MiniLisp and LRacer API are introduced to support the student in his or her task to develop various evaluation metrics, integrated in the RacerPro environment. The chapter makes the student aware of some worst practices that occur during ontology engineering.

9.1 Dimensions of ontology evaluation

When ranking ontologies, the challenge is to take into consideration all their semantic, syntactical, and contextual aspects. The ranking methods from the literature focus on evaluating specific parts of ontologies. There is still need of a new ranking algorithm that should take into account more aspects of the evaluation. For instance, AktiveRank [1] is a technique for ranking ontologies based on the analysis of concept structures, while the OS_Rank [55] algorithm ranks ontologies based on semantic relations and structure. These two methods use SWOOGLE [12] for searching the ontologies that match some terms from the user queries.

Another type of ranking approach is based on popularity, measured in terms of referrals and number of citations between ontologies. Such a method is defined by the semantic search engines like SWOOGLE [12] and OntoKhoj [40] that use PageRank algorithm to rank ontologies. Due to the fact that ontologies are not so well connected and cited like web pages are, this ranking method may be not so efficient if applied on ontologies. Several ontology evaluation metrics have been proposed by OntoQA [52], which also applies a scoring-based ranking method on the metric scores results.

9.2 Ontology evaluation metrics

For the structural evaluation, the metrics indicate how well the ontologies were designed with respect to the size, depth, width, density, richness, inheritance of a schema. The semantic layer of the evaluation is based on instance metrics that check how the instances were placed in the ontology and the usage of the real-world knowledge representation. The following metrics definitions are adapted from [52, 1, 55].

Structural Evaluation

The metrics related to the structural layer are also known as schema metrics and they analyze ontologies as graph structures. This kind of evaluation does not verify if the knowledge is correctly modeled in the ontology.

In structural evaluation, ontologies are analysed as graph structures.

- size (how many concepts, roles, individuals)
- depth and breadth of hierarchy
- density (average branching)
- balance (are all area equally developed?)
- overall complexity
- connectivity between concepts, relations (highly interconnected, loosely interconnected, e g. only by isa-links?)
- richness of relationships, attributes, inheritances (isa, type-of, subclass, part-of, instance-of, multiple inheritance).
- Reasoning (required tasks)
- Modularization (what modules are defined? How are they defined? How is re-use, import, export of modules regulated, if at all?).

The main idea behind ontology modularization is the identification of reusable knowledge. A module represents a shared, domain-independent conceptualization intended to be used for different tasks and applications. From this perspective, ontology modules are comparable to software libraries in the software engineering domain.

Attribute Richness (AR). represents the ratio between the total number of attributes A in the ontology and the number of concepts C. The metric indicates how much knowledge about classes is provided by the ontology on average.

$$AR = \frac{\mid A \mid}{\mid C \mid} \tag{9.1}$$

Relationship Richness (RR). is the ratio between the total number of roles R in the schema and the sum of the number of subclasses SC plus the number of roles R. The metric indicates which is the percentage of rich relationships defined from the total number of relationships in the schema (rich and inheritance relationships).

$$RR = \frac{\mid R \mid}{\mid SC \mid + \mid R \mid} \tag{9.2}$$

Inheritance Richness (IR). represents the ratio between the total number of subclasses SC and the total number of concepts C defined in the schema. The metric aims to identify if the ontology has a vertical or an horizontal structure.

$$IR = \frac{\mid SC \mid}{\mid C \mid} \tag{9.3}$$

Semantic Evaluation

Semantic evaluation checks the amount of the real-world knowledge represented by the ontologies and how well the instances are placed and distributed within the schema. The metrics for this type of evaluation can be grouped in two categories [52]: knowledge base metrics and class metrics. Another part of the semantic evaluation is represented by the competency questions module. It intends to evaluate how well the domain was modeled by interrogating the ontologies with some domain-oriented questions. Depending on the number of answers an ontology provides, we can determine which are the ontologies most representative for certain domains.

Class Richness (CR) is defined as the ratio of the number of non-empty concepts used in the schema C' to the total number of concepts C in the ontology. The metric evaluates the distribution of instances across the concepts defined, showing the percentage of the data used in the KB schema representation.

$$CR = \frac{\mid C' \mid}{\mid C \mid} \tag{9.4}$$

Average Population (AP) is the total number of instances I in the schema divided by the number of concepts C. The metric checks how well the instances were distributed across the classes of the ontology.

$$AP = \frac{\mid I \mid}{\mid C \mid} \tag{9.5}$$

Cohesion (Coh) is represented by the total number of separated connected components SCC of the graph in the KB. It indicates which are the areas in which instances could be more closely connected.

$$Coh = \mid SCC \mid \tag{9.6}$$

Importance (Imp) is defined as the ratio between the total number of instances that belong to the subtree rooted at the current class (C_i(I)) to the total number of instances I in the schema. The metric shows the distribution of instances over classes and identifies which are the areas in focus towards the instances extraction process.

$$Imp_i = \frac{\mid C_i(I) \mid}{\mid I \mid} \tag{9.7}$$

Inheritance Richness for a Concept (IRC) represents the number of subcocenpts in the subtree starting from the current concept ($SC(C_i)$) to the total number of nodes N in the subtree .

$$IRC_i = \frac{\mid SC(C_i) \mid}{\mid N \mid} \tag{9.8}$$

Relationship Richness for a Concept (RRC). represent the number of relationships to which the instances of the current concept are connected to the total number of relationships defined for the current concept. The metric indicates how many properties of a certain class are used at instances level.

$$RRC_i = \frac{\mid R(I_i, I_j) \mid}{\mid R(C_i) \mid}, I_i \in C_i, I_j \in C \tag{9.9}$$

Connectivity (Con). defines the number of instances of other concepts (I_j) connected to the instances of the concept being evaluated ($C_i(I)$). This metric indicates which classes play a central role depending on the popularity of instances in the schema.

$$Con_i =\mid I_j, R(I_i, I_j \wedge I_i \in C_i(I)) \mid \tag{9.10}$$

9.3 MiniLisp

The RacerPro server can be programmed in the *miniLisp* functional language. Consider the following gentle introduction:

```
? (evaluate (+ 2 3))
> 5
? (evaluate (format t "I␣am␣a␣string"))
  I am a string
> NIL
? (evaluate (format t "The␣value␣is:␣~A" (+ 4 3))
  The value is: 7
> NIL
? (evaluate (first '(a b c)))
> A
? (evaluate (dotimes (i 3) (format t "Place␣~A!" (+ 1 i))))
  Place 1! Place 2! Place 3!
> NIL
? (define plus (a b) (+ a b))
> plus
? (evaluate (plus 2 3))
> 5
```

Note that the miniLisp functions can be combined with the RacerPro commands:

```
? (evaluate (first (all-atomic-concepts)))
> TOP
```

The available miniLisp functions can be found at http://localhost:8080/minilisp.html. Note that macros in RacerPro come with their corresponding functions. The main difference here is that the arguments of a functions are evaluated.

```
(define-concrete-domain-attribute width :type integer)
(define-concrete-domain-attribute length :type integer)
(instance rectangle1 (and (equal width 10) (equal length 20)))

(evaluate (add-concept-assertion 'RACER::DEFAULT 'rectangle2
                  `(and (equal width ,(+ 2 3))
                        (equal length ,(* 2 3)))))
```

You can check that the computations were performed by describing the individual `rectangle2`:

```
> describe-individual rectangle2)
> (rectangle2
      :SYNONYMS (rectangle2)
      :ANTONYMS NIL
      :ASSERTIONS ((rectangle2 (AND (EQUAL width 5) (EQUAL LENGTH 6))))
      :ROLE-FILLERS NIL
      :TOLD-ATTRIBUTE-FILLERS NIL
      :TOLD-DATATYPE-FILLERS NIL
```

```
9         :ANNOTATION-DATATYPE-PROPERTY-FILLERS NIL
10        :ANNOTATION-PROPERTY-FILLERS NIL
11        :DIRECT-TYPES :TO-BE-COMPUTED)
```

Full access to the LISP environment is supported thought the LRacer API.

LRacer. `LRacer` is the API for Common Lisp to access all functions and macros provided by RacerPro. Assuming that `LRacer` is in the directory `LRACER_HOME`, start a common lisp implementation (i.e. clisp) and evaluate:

```
1 (load "LRACER\_HOME/lracer-sysdcl.lisp")
2 (compile-load-racer "LRACER\_HOME/").
```

The ontology evaluation metrics can be implemented in LRacer as follows:
Metric: Number of Concepts (NOC)

```
1 (defun noConcepts (tbox)
2  (length (all-atomic-concepts tbox)))
```

Metric: Number of Roles (NOR)

Listing 9.1: Number of roles in the ontology

```
1 (defun noRoles (tbox)
2  (length (all-roles tbox)))
```

Metric: Number of Individuals (NOI)

Listing 9.2: Number of ind in the ontology

```
1 (defun noRoles (tbox)
2  (length (all-roles tbox)))
```

Semantic layer

Ontologies can also be evaluated considering the way data is placed within the ontology or in other words, the amount of real-world knowledge represented by the ontology. These metrics are refereed to as knowledge base metrics and include [31]:

- Consistency
- Expressiveness (ontology language used)
- Comprehensiveness (extent of target domain covered, e.g. answering to competency questions).

Description logics provide different expressive power. The minimal terminological language, called $\mathcal{AL}$ (attributive language), enables intersection of concepts, negation of atomic concepts, restriction on values and limited existential quantification. The following expressiveness can be added to $\mathcal{AL}$ language:

$$\mathcal{AL}[U][\varepsilon][N][C][H][R]$$

where: U denotes union of concepts, ε, full existential quantification N - number restrictions, C - negation of complex concepts H - role hierarchies R - intersection of roles R^+ - transitive roles

```
(get-tbox-language)
(get-abox-language)
```

Number of cycles is a measure of the ontology consistency.
Metric: Number of cycles

Listing 9.3: Number of cycles in the ontology

```
(defun noCycleConcepts (tbox)
 (length (tbox-cyclic-p tbox)))
```

9.4 Worst practices and Anti-patterns

ODPs ideally represent "best practices". Worst practices are solutions unsuitable for the designing ontologies. A number of 231 such worst practices were identified in [46]. Part of them are listed in Table 9.4.

Table 9.1: Worst practices description.

Bad Practice	Description
Relationship "is"	Confusion with subclass relationship, membership of o class or same as
Recursive definition	Using an ontology element in its own definition.
Undefined inverse relationships	Having inverse relationships in the ontology, but they are not explicitly defined as such.
Lazy elements	Leaf concepts or roles that never appear in the application and do not have any instances.
Missing disjointness	Lack of disjoint axioms between classes or between properties that should be defined as disjoint (even-odd, prime-composite)
Individuals are not Classes	For instance, `Madrid` is an instance not a concept
Redundancy of Disjoint Relation	For instance, the concept is explicitly defined as disjoint with parent concept and also with its child concept.
Missing subclasses	Number of classes which have only one subclass. The situation indicates that either: i) the hierarchy is under-specified or ii) the distinction between the subclass is not appropriate.
Extra subclasses	Number of classes with more than 25 subclasses. Such a class is candidate for additional distinctions

Anti-patterns do exist in the field of ontology engineering. For instance, *Redundancy of Disjoint Relation* anti-pattern encapsulates the modelling error: a concept is disjoint with any concept then it is also disjoint with its sub concepts.

```
(disjoint male female)
(implies woman female)
(disjoint male female)
```

For more examples of anti-patterns, to curious reader is referred to [9]

9.5 Exercises

Exercise 9.1 (Ontology evaluation metrics)**.**

- *Translate your favorite ontology from KRSS syntax into OWL format*
- *Evaluate your favorite ontology using the metrics provided by OWLAPI. You can use http://owl.cs.manchester.ac.uk/repository.*
- *Browse the code provided by the OWLAPI with the goal to identify the metrics provided by the OWLAPI library.*

Exercise 9.2 (Family of description logics)**.** *Draw a partial order diagram displaying syntactic containment of all DLs that match the above naming scheme and do not contain* $\mathcal{F}$ *or* $\mathcal{N}$*.*

Exercise 9.3 (Family of description logics)**.** *Name, for each of the following knowledge bases, the "smallest" DL that contains it:*

- *The* `Family` *ontology from exercise*
- *The* `Bibtex` *ontology*
- *The knowledge base containing the axiom:*

```
1 (implies batman (not (some (inv similar) (one-of  santa)))))
```

Solution

Exercise 9.4 (Modelling in DL)**.** *Define a small ontology of our solar system. Display an interpretation as a directed graph with labeled nodes and arcs.*

9.6 Solutions

Solution 9.1 (Exercise 9.2)**.** *Use (racer-read-file " /file.racer") to load the corresponding ontology. then, you may check the active tbox and ask for the description logic used in the current tbox: For checking the last ontology, define a new TBox containing the two axioms given.*

```
1 ? (get-tbox-language)
2 > (ONE-OF santa) encoded as (OR santa)
3 > "FLO"
```

Chapter 10

Documenting Ontology

This chapter has two objectives. The first objective is to introduce the most common annotation properties used to document the ontology. The second one is to introduce the technical instrumentation which automatically generates ontology documentation. We focus on HTML and Latex formats.

10.1 Annotation properties

Elements of an ontology can be annotated with metadata. Several annotation properties (i.e., `owl:versionInfo`) are predefined and users can define various annotation properties. The documentation of an ontology can start from labels (i.e., `rdfs:label`), comments (i.e., `rdfs:comment`), and other kinds of annotations (i.e., `dc:description`, `dc:creator`, `dc:date`).

The annotation properties are grouped into metadata vocabularies: Dublic Core[1], RDF Schema[2], annotation properties of OWL[3], SKOS[4], VANN[5], FOAF[6].

Dublin Core provides metadata elements for annotating documents [10], as one of the most popular vocabulary for use with RDF and recently with Linked Data movement. Some of its annotation properties are listed in Table 10.2. RDF Schema (RDFS) includes basic annotation properties such as `rdfs:label` and `rdfs:comment`, `rdfs:seeAlso` or `rdfs:isDefinedBy` (see Table 10.2). OWL introduces annotation roles for capturing version information and compatibility notes. As OWL is built on top of RDFS, the use of rdfs annotation properties is encouraged. The Simple Knowledge Organization System (SKOS) is a RDF-based vocabulary for expressing the basic structure and content of concept schemes (thesauri, classification schemes, taxonomies, terminologies, and other types of controlled vocabulary) [35]. The Friend of a Friend (FOAF) ontology is intended to describe persons, their activities and their relations with other people and objects [6]. However, in practice some of its elements are used as annotation properties (i.e., foaf:maker). Each term in FOAF is annotated with properties from the SemWeb Vocab Status Ontology[7].

[1] http://www.dublincore.org/documents/dces/
[2] http://www.w3.org/TR/rdf-schema/
[3] http://www.w3.org/TR/owl-ref/#Annotations
[4] http://www.w3.org/2004/02/skos/
[5] http://vocab.org/vann/.html
[6] http://www.foaf-project.org/
[7] http://www.w3.org/2003/06/sw-vocab-status/note

Table 10.1: Tools generating HTML documentation of an ontology.

Tool	Feature	Available
OWLDoc	Generates JavaDoc-like HTML	http://protegewiki.stanford.edu/wiki/OWLDoc
Neologism	Web based ontology management system	http://neologism.deri.ie
VocDoc	Ruby script for documenting RDFS/OWL ontologies. It also generates LATEX reports	http://kantenwerk.org/vocdoc
SpecGen	Produces ontology specification	http://forge.morfeo-project.org/wiki_en/index.php/ SpecGen
LODE	Online documentation service that considers both axioms and annotations	http://www.essepuntato.it/lode
Parrot	Multilingual online documentation tool that collects annotations in a RDF graph	ontorule-project.eu/parrot

The RacerPro command `(all-annotation-concept-assertions)` returns the values for the identified annotation properties in an ontology. The `owlapi-keep-annotations` function can be exploited to access annotations from RacerPro. The opposite function is `owlapi-ignore-annotations`. You can also investigate `OWLAPI-writeOntologyFile` function to include comments in your ontology. To obtain the annotation for a concept, investigate the function `OWLAPI-getOWLAnnotationPropertyDomainAxiom`. The function gets as input an annotation property (e.g. `rdfs:comment`) and a RacerPro concept (e.g. `(some has-child Person)`). To obtain annotations about an individual, the function `OWLAPI-getOWLAnnotationAssertionAxiom` can be called with an `annotation-subject` (name of an individual, i.e., `alice`) and the annotation role (`rdfs:comment`).

10.2 Visualising ontologies in the HTML format

Among various available tools for generating HTML documentation of an ontology (see Table10.1), this section exemplifies two such tools: LODE[8] and Parrot[9].

The Live OWL Documentation Environment (LODE) automatically generates a human-readable description of an OWL ontology, taking into account both ontological axioms and annotations [42]. LODE extracts classes, object properties, data properties, named individuals, annotation properties, general axioms, SWRL rules, and namespace declarations from OWL or OWL2 ontologies. The input is the RDF/XML linearisation of an ontology produced through the OWLAPI. The output is an HTML representation of the ontology.

The Parrot documentation service [53] also exploits the metadata annotation when generation documentation. The system lets the user to choose the language used for documentation. It generates reports with the flavors: business oriented and technically oriented. The tool can be used as a Web service, as an Eclipse plugin, and as a command line interface.

[8]http://www.essepuntato.it/lode

[9]http://ontorule-project.eu/parrot/parrot

Consider the following ontology in the tourism domain.

```
(define-concept TourismAxis (or Accomodation Activity Gastro POI))
(disjoint Accomodation Activity)
(disjoint Accomodation Gastro)
(disjoint Accomodation POI)
(disjoint Activity Gastro)
(disjoint Activity POI)
(disjoint Gastro POI)

(define-primitive-role hasAccomodation
    :domain (or Activity  Gastro POI) :range Accomodation)
(define-primitive-role hasActivity
     :domain (or Accomodation Gastro POI) :range Activity)
(define-primitive-role hasEatingAndDrinking
     :domain (or Accomodation Activity POI) :range Gastro)
(define-primitive-role hasPOI
     :domain (or Accomodation Activity Gastro) :range POI)
(define-primitive-role hasLocation
     :domain TourismAxis :range Location
     :transitive t :inverse isLocatedIn)

(instance DraculaInn Accomodation)
(related DraculaInn Transilvania isLocatedIn)
```

A solution to generate the documentation with the Parrot documentation service is:

1. Using RacerPro save the ontology in the OWL format.
2. Using the OWL Manchester converter, translate it into turtle format.
3. Load the ontology in the turtle format to the Parrot service by direct input (ontoruleproject.eu/parrot/)
4. Specify the input format (turtle) and generate technical report.

The upper part of the report summarises the concepts, roles and individuals in the ontology (Fig. 10.2). The bottom part of the Fig. 10.2 details the role `isLocatedIn`. The role is identified as transitive, its inverse is specified (`hasLocation`), and the domain and range are also visible in the report.

10.3 Documenting ontologies in the Latex format

Consider the tourism ontology introduced in section 10.2. One possibility to obtain the latex documentation of the ontology is to apply the following steps:

1. Load the ontology in RacerPro: `(racer-read-file "tourism.racer")`
2. Save the ontology in OWL syntax: `(save-kb "tourism.owl" :syntax :owl)`
3. Convert the ontology in .tex: go to the OWL syntax converter at http://mowlpower.cs.man.ac.uk:8080/converter/. Paste your ontology and select the latex format for conversion.
4. Convert tex sources into dvi: run the command `latex tourism.tex`.

.

Figure 10.1: Part of HTML report generated by the Parrot documentation service.

5. Convert dvi into pdf: run the command `dvipdf tourism.dvi tourism.pdf`.

The resulted documentation in the pdf format is illustrated in Fig. 10.2.

TOURISMAXIS

TOURISMAXIS ≡ ACCOMODATION ⊔ ACTIVITY ⊔ GASTRO ⊔ POI

Object properties

HASACCOMODATION

∃ HASACCOMODATION Thing ⊑ ACTIVITY ⊔ GASTRO ⊔ POI
⊤ ⊑ ∀ HASACCOMODATION ACCOMODATION

Figure 10.2: Part of the Latex documentation in DL format.

10.4 Exercises

Exercise 10.1 (Annotation properties)**.** *Load the bibtex ontology from the web page http://zeitkunst.org/bibtex/0.2/bibtex.owl. Identify the values of the following annotation properties:* `dc:creator`*,* `dc:description`*,* `dc:date`*,* `dc:subject`*,* `rdfs:label`*,* `rdfs:comment` *and* `owl:versionInfo`*. Solution*

Exercise 10.2 (Annotation properties). *Some of the annotation properties in the common metadata vocabularies can overlap. For the following questions, formalise your answer in RacerPro:*

1. *Check if the* `vann:changes`, `vann:example`, *and* `vann:usageNote` *are subroles of* `rdfs:seeAlso`.
2. *Is the* `dc:creator` *role equivalent to the role* `foaf:maker`*?*

Exercise 10.3 (Annotation properties). *Consider the following popular web vocabularies: dbpedia, foaf, geonames, skos, geo, bibo.*

1. *Check which specified vocabularies use the following annotation properties:* `rdfs:label`, `rdfs:comment`, `rdfs:isDefinedBy`, `rdfs:seeAlso`, `dc:title`, `dc:description`, `owl:versionInfo`, `dc:date`, `dc:creator`.
2. *Sort the annotation properties in decreasing frequency of use. Build a table with vocabularies as columns and annotation properties as lines.*

Solution

Exercise 10.4 (Annotation properties). *Consider the ODP from the web page http://www.ontologydesignpatterns.org.*

1. *Using RacerPro identify and analyse the annotation properties*
2. *Generate HTML documentation of your preffered ODP*

Exercise 10.5 (Visualising ontologies in HTML). *Using LODE, explore the FOAF ontology from http://xmlns.com/foaf/spec/index.rdf.*

Exercise 10.6 (Visualising ontologies in HTML). *Note that the Parrot documentation service was able to identify only one individual in the Tourism ontology -* `DraculaInn` *which was explicitly stated by* `(instance DraculaInn)`*. Instead, RacerPro is aware of two individuals, deduced from the argument type of the* `related` *macro.*

```
1 ? (all-individuals)
2 > (DRACULAINN TRANSILVANIA)
```

Try to syncronise the documentation generated by Parrot with the actual semantics of the ontology.

Exercise 10.7 (Understanding ontologies). *Consider the FRBR-aligned Bibliographic Ontology (FaBiO) from http://purl.org/spar/fabio.*

1. *Identify how many concepts, roles, attributes, and individuals the ontology contain?*
2. *Describe the main aim of the ontology.*
3. *Describe what the concept* `DoctoralThesis` *defines.*
4. *Describe what the role* `hasSubjectTerm` *defines, and identify its domain and range concepts.*

5. *Identify the class having the largest number of direct individuals. (Recall that a direct individual belongs explicitly to that concept and is not inferable from its subconcepts).*

6. *List all the subconcepts and roles involving the concept Item.*

Solution

Exercise 10.8 (Ontology visualisation)**.** *Consider the following addition ontology visualisation method:*

1. *Graphical visualisation - Obtain a graphical representation of the Pizza ontology.*

2. *UML - Construct the UML diagrams of the Pizza ontology.*

3. *Natural language: Identify a tool that claims to translate description logic statements into natural language. Apply the tool on the Pizza and Tourism ontology. Propose a method to integrate this tool with the RacerPro reasoning engine.*

Solution

Exercise 10.9 (Use cases)**.** *Describe use cases and application scenarios of your preferred ontology. Solution*

10.5 Solutions

Solution 10.1 (Exercise 10.1)**.** *O first solution to retrieve all the annotation properties is:*

```
? (OWLAPI-readOntology "http://zeitkunst.org/bibtex/0.2/bibtex.owl"
        :MAINTAIN-OWLAPI-AXIOMS T
        :kb-name book-shop)
> book-shop
?  (all-annotation-concept-assertions)
> ((#!bibtex:pageChapterData
      (D-FILLER #!rdfs:comment
      (D-LITERAL "A␣generic␣property␣to␣hold␣page␣and/or␣chapter␣data."
      (D-BASE-TYPE #!xsd:string))))
  ......................
```

A second solution would be to exploit the **Assertions** *tab of RacerPorter and to select to annotation properties for all concepts.*

Solution 10.2 (Exercise 10.7)**.**

1. *The FaBiO ontology has 214 concepts, 69 roles, 45 attributes, and 15 individuals.*

2. *The aim is to describe bibliographic entities such as research papers, journal articles and books.*

Solution 10.3 (Exercise 10.3)**.** *1. A solution would be to generate the ontology documentation with the LODE framework. As an example, the FOAF Vocabulary Specification 0.99* [10] *has the following annotation properties:* **wot:assurance**, **dc:date**, **dc:description**, **foaf:membershipClass**, **foaf:name**, **wot:src_assurance**, *and* **dc:title**. *Fig. 10.3 bears out these annotation properties. The parameters used by LODE were: 'OWLAPI + Imported', respectively 'Use the reasoner'*

[10] The namespace document from 14 January 2014 available at http://xmlns.com/foaf/spec/.

2. *A similar analysis is presented in [53] for a larger set of vocabularies and annotation properties.*

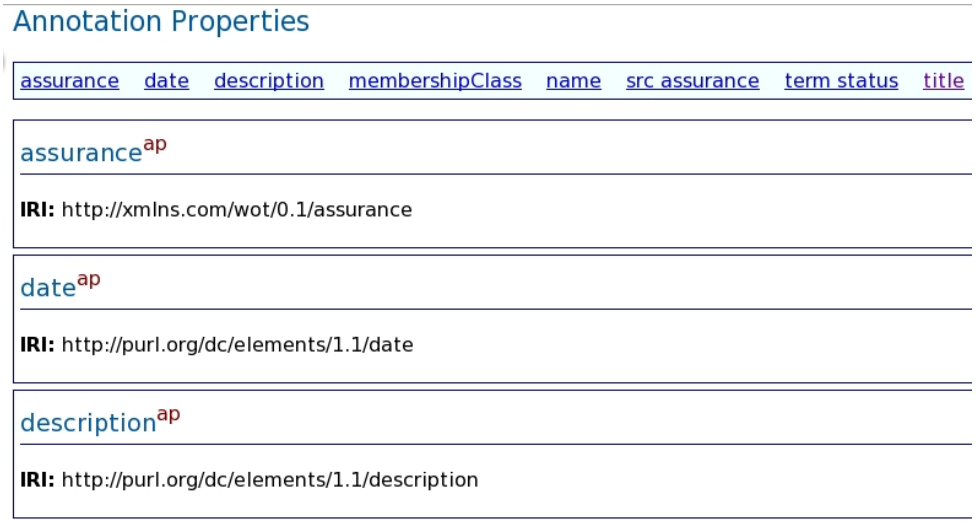

Figure 10.3: LODE depicts annotation properties of the FOAF ontology.

Solution 10.4 (Exercise 10.8)**.**

1. *Article [30] presents a survey of the available instrumentation for ontology visualisation.*
2. *A visual UML-based notation for ontologies is introduced in [7].*
3. *NaturalOWL [15] can be used to generate natural language from OWL statements.*

Solution 10.5 (Exercise 10.9)**.** *See one example at http://ontologies.hypios.com/.*

Table 10.2: Annotation properties used for the description of ontologies.

Annotation	Description
dc:contributor	An entity responsible for making contributions to the ontology. Examples of a Contributor include: a person, an organization, or a service.
dc:creator	An entity primarily responsible for making the resource.
dc:date	A point or period of time associated with an event in the lifecycle of the ontology. Use the encoding scheme W3CDTF profile of ISO 8601.
dc:description	An account of the resource such as: an abstract, a table of contents, a graphical representation, or a free-text account of the resource.
dc:rights	Information about rights held in and over the ontology, including intellectual property rights.
dc:title	A name by which the resource is formally known.
dc:subject	The topic of the resource. It is represented using keywords or classification codes. Recommended best practice is to use a controlled vocabulary.
rdfs:label	Provides a human-readable version of a resource's name.
rdfs:comment	Provide a human-readable description of a resource.
rdfs:seeAlso	Indicate a resource that might provide additional information about the subject resource.
rdfs:isDefinedBy	Used to indicate a vocabulary in which a resource is described.
owl:priorVersion	Used to track the version history of an ontology.
owl:versionInfo	Used to annotate version information at the granularity of concepts and roles.
skos:prefLabel	Used to assert the preferred lexical label for a resource.
skos:altLabel	Used any number of times to assert alternative lexical labels for a concept.
skos:definition	Used to assert a definition for the meaning of the given resource.
vann:changes	A reference to a resource that describes changes between this version of a vocabulary and the previous.
vann:example	A reference to a resource that provides an example of how this resource can be used
vann:usageNote	A reference to a resource that provides information on how this resource is to be used.
foaf:maker	An agent that make a thing.
vs:term_status	Status of a vocabulary term is one of 'stable','unstable','testing' or 'archaic'.

Part II

Applications

Chapter 11

Engineering a Romanian Tourism Ontology

This chapter presents a large domain ontology for the Romanian tourism. The ontology was developed in the KRSS syntax in a modular way. After introducing the core ontology, the chapter details various modules that encapsulate knowledge on: accommodation types and their facilities, touristic activities, points of interest or eating and drinking resorts. For populating the ontology various sources were linked like Foursquare, Open StreeMap or the Open Linked Data for Romania.

11.1 Core ontology

Fig. 11.1 illustrates the main four classes of the Romanian tourism ontology as well as the relationship between them materialized into roles `hasActivity`, `hasAccomodation`, `hasEatingAndDrinking` and `hasPointOfInterest`. The spatial dimension is attached through the `hasLocation` role, between the four concepts and the `Location` concept (see Fig. 11.1).

```
1  (define-concept TourismAxis (or Accommodation Activity
2                                   Gastro POI Location))
3  (disjoint Accommodation Activity Gastro POI Location)
4  (define-role hasAccommodation :domain (or Activity Gastro POI)
5                                :range Accommodation)
6  (define-role hasActivity :domain (or Accommodation Gastro POI)
7                           :range Activity)
8  (define-role hasEatDrink :domain (or Accommodation Activity POI)
9                           :range Gastro)
10 (define-role hasPOI :domain (or Accommodation Activity Gastro)
11                     :range POI)
12 (define-role hasLocation :domain TourismAxis
13                          :range Location
14                          :transitive t
15                          :inverse LocatedIn)
```

Figure 11.1: Top level concepts in the tourism ontology.

Beside the top level concepts, we also introduced the concept `Match` for representing the relations between the touristic places and the blog posts that POIs appeared in. This

```
1  (define-role fromBlogPost :domain Match :range BlogPost)
2  (define-role hasSubject :domain Match :range TourismAxis)
3  (define-concrete-domain-attribute hasScore :domain Match :type real)
4  (define-concrete-domain-attribute hasText :domain Match :type string)
5  (define-role speaksAbout :domain BlogPost :range TourismAxis)
6  (instance m1 Match)
7  (instance b100 BlogPost)
8  (instance mateicorvin POI)
9  (attribute-filler m1 "casa␣matei␣corvin␣atrage␣multi␣turisti" hasText)
10 (related m1 b100 fromBlogPost)
11 (related m1 mateicorvin hasSubject)
12 (attribute-filler m1 0.8)
```

Figure 11.2: Relating information about a blog with the *n-ary design* pattern.

```
1  (instance InternetAccess Facility)  (instance AirConditioning Facility)
2  (instance Golf Facility)            (instance Restaurant Facility)
3  (instance Babysitting Facility)     (instance Gym-fitness Facility)
4  (instance Casino Facility)          (instance KidsClub Facility)
5  (instance Beachaccess Facility)     (instance Sofabed Facility)
6  (instance Washingmachine Facility)  (instance Parking Facility)
7  (instance Refrigerator Facility)    (instance Dryer Facility)
8  (instance Living-lounge Facility)   (instance Iron Facility)
9  (instance Changeofthelinen Facility)(instance Terrace-Balcony Facility)
10 (instance Diningroom Facility)      (instance Weeklycleaning Facility)
```

Figure 11.3: Sample from the 200 facilities formalised in the LELA ontology.

concept was modelled by enacting the *n-ary ontology design pattern* [48]. The goal was to combine several information about a tourism blog regarding: subject of the blog according to the concepts in the ontology, computed score about an instance in the ontology, or provenance information like author, starting and ending text index (text position) which relates to an identified instance in our ontology.

As an example, the individual `m1` of type `Match` is related to the blog `b100` via the role `fromBlogPost`. The point of interest `mateicorvin` is related to the same match `m1` by the relation `hasSubject`. The positive score of 0.8 in line 22 is computed with a basic opinion mining algorithm from the blog post.

11.2 Modelling knowledge on accommodation

Each accommodation provides various facilities. The corresponding TBox contains a list of 200 facilities. Fig. 11.3 presents 20 of these facilities.

Each accommodation type is located in a city. The ontology uses a list of 428 cities in Romania. Ten cities in alphabetical order are listed in Fig. 11.4.

We use two Aboxes for asserting facts about accommodation. The first Abox includes 555 individuals of type `Accommodation`. These instances are categorized in the following categories: `GoodHotel`, `LuxuryHotel`, `ExtraLuxuryHotel` or `Budget` accomodation, according to the number of stars (see Fig. 11.5).

A booking is encapsulated in the ontology as:

```
1  (instance b1 Booking (string= has-start-date "2014-05-13")
```

```
(instance Vacaresti-Dimbovita City)    (instance VaduIzei City)
(instance Vaduri City)                 (instance ValeaLuimihai City)
(instance ValeaLunga City)             (instance ValeniiDeMunte City)
(instance VamaVeche City)              (instance Vanatori City)
(instance Vanatorii-Mari City)         (instance Varsand City)
(instance Vaslui City)                 (instance VatraDornei City)
(instance Vernesti City)               (instance Vicsani City)
(instance Victoria City)               (instance Videle City)
(instance Vidra-Vrancea City)          (instance Vinga City)
(instance ViseuDeSus City)             (instance Vladesti City)
(instance Vladimirescu City)           (instance Vlahita City)
```

Figure 11.4: Sample from the 428 cities used to locate various accommodation types.

```
(implies GoodHotel (and Hotel (has-value hasRating ThreeStarRating)))
(implies LuxuryHotel (and Hotel (has-value hasRating FourStarRating)))
(implies ExtraLuxuryHotel (and Hotel
                               (has-value hasRating FiveStarRating)))
(implies Campground (and Accomodation
                         (has-value hasRating OneStarRating)))
(implies Hostel (and Accommodation (all hasRating TwoStarRating)))
(implies Chalet (and Accommodation (all hasRating ThreeStarRating)))
(define-concept BudgetAccommodation (and Accommodation (some hasRating
                                                       (or OneStarRating
                                                           TwoStarRating))))
```

Figure 11.5: Formalising hotel classifications.

```
(string= has-end-date "2014-06-15") (= has-price 130.25))
```

Some hotels in the ontology are exemplified in Fig. 11.6.

For each hotel, the provided facilities are specified. Fig. 11.7 depicts the facilities offered by the `RamadaPlazaBucharestConventionCenter`.

The second ABox contains 2517 hotels, with related information described by the following features: `name, has-address, has-phone, has-website` (see Fig. 11.8).

11.3 Modelling touristic activities

We consider several types of tourism, as defined in Fig. 11.9.

We also defined 61 touristic-related activities. Fig. 11.3 lists a sample of 20 such activities.

Activities around a specific accomodation are stored as assertions in Fig. 11.3. The query `(individual-fillers IoanaPension doActivity)` retrieves all known activities at the `IoanaPension` resort.

The ontology contains a taxonomy of 80 geographical touristic objectives, as illustrated by Fig. 11.12.

The corresponding TBox for these geographical features is also provided. Part of this taxonomy is presented in Fig. 11.13.

Information about administrative areas is stored in terms of `region, country, city and 462 communes`, as Fig. 11.14 illustrates.

```
(instance CETATE GoodHotel)
(instance MA GoodHotel)
(instance Europa GoodHotel)
(instance ContinentalArad LuxuryHotel)
(instance BestWesternCentralHotel GoodHotel)
(instance ImparatulRomanilorParc GoodHotel)
(instance Parc GoodHotel)
(instance Phoenix GoodHotel)
(instance President GoodHotel)
(instance HotelCoandi GoodHotel)
(instance ARARATHOTEL GoodHotel)
(instance Decebal BudgetAccommodation)
(instance Moldova GoodHotel)
(instance SunGardenGolf&SpaResort ExtraLuxuryHotel)
(instance Carpati LuxuryHotel)
(instance BestWesternEurohotel GoodHotel)
(instance Maramures GoodHotel)
(instance Mara GoodHotel)
(instance BestWesternHotelEurohotel GoodHotel)
(instance RivulusBaiaMare GoodHotel)
```

Figure 11.6: Sample from the 555 classified hotels in the LELA ontology.

11.4 Modelling points of interest

The tourism ontology includes relevant information about 637 museums in Romania. A sample of these museums is listed in Fig. 11.15.

Each museum has a textual description in Romanian, using the role `has-description`

`(related |denumirea (romana)| |descrierea (romana)| has-description )`

Each museum has the following features: `has-location`, `has-email`, `has-phone`, `has-founding-year`, `has-latitude`, `has-longitude`, `county-has-museum`, `has-schedule`, `has-website`. The exemplification of these features appears in Fig. 11.4.

The correspondence between Romanian name and the English one is handled by the `same-as` operator (see Fig. 11.17).

We focuses also on modelling points of interests like caves. The ontology contains data on 953 caves in Romania, as Fig. 11.18 exemplifies.

Each cave has a specifiec location, as exemplified by the following RacerPro code:

```
(related |CLUJ| |Pestera Zanelor| has-cave)
(related |Muntii Trascau| |Pestera Zanelor| has-cave)
(related |BIHOR| |Pestera de la Zapodie| has-cave)
```

The concept `Mountain` is instantiating with the corresponding individuals in Romania (see Fig. 11.19) In the same figure, each mountain belongs to a particular county.

The ontology contains also 729 instances of the concept `POI` (see Fig. 11.20).

11.5 Populating the ontology

Importing Open Street Map

Data from Open Street Map (OSM) can be directly integrated as an ABox in the tourism ontology. For converting OSM into racer syntax we developed a java-based converter

```
(same-as RamadaPlazaBucharestConventionCenter RamadaPBCC)
(related RamadaPBCC InternetAccess has-facility)
(related RamadaPBCC AirConditioning has-facility)
(related RamadaPBCC Restaurant has-facility)
(related RamadaPBCC Gym-fitness has-facility)
(related RamadaPBCC KidsClub has-facility)
(related RamadaPBCC Parking has-facility)
(related RamadaPBCC Refrigerator has-facility)
(related RamadaPBCC Terrace-Balcony has-facility)
(related RamadaPBCC Diningroom has-facility)
(related RamadaPBCC Tenniscourt has-facility)
(related RamadaPBCC Oven-microwave has-facility)
(related RamadaPBCC TV has-facility)
(related RamadaPBCC Establishmentadaptedfordisabled has-facility)
(related RamadaPBCC Businesscenter has-facility)
(related RamadaPBCC Hairdressingsalon has-facility)
(related RamadaPBCC RoomService has-facility)
(related RamadaPBCC Laundry has-facility)
(related RamadaPBCC Mini-bar has-facility)
(related RamadaPBCC Safeinroom has-facility)
(related RamadaPBCC swimmingpool has-facility)
(related RamadaPBCC ValetParking has-facility)
(related RamadaPBCC Bar has-facility)
(related RamadaPBCC Spa-Aesthetic has-facility)
(related RamadaPBCC Elevator has-facility)
(related RamadaPBCC Meeting-conference has-facility)
(related RamadaPBCC Wake-upService has-facility)
(related RamadaPBCC Suites has-facility)
(related RamadaPBCC Yearofconstruction has-facility)
(related RamadaPBCC Lastrenovated has-facility)
(related RamadaPBCC Numberoffloorsannexbuilding has-facility)
(related RamadaPBCC Totalnumberofrooms has-facility)
(related RamadaPBCC Singlerooms has-facility)
(related RamadaPBCC Double-twinrooms has-facility)
(related RamadaPBCC MasterCard has-facility)
(related RamadaPBCC Visa has-facility)
(related RamadaPBCC Publictransport has-facility)
(related RamadaPBCC Bathroom has-facility)
(related RamadaPBCC Directdialtelephone has-facility)
(related RamadaPBCC Breakfastbuffet has-facility)
```

Figure 11.7: Browsing facilities for the Ramada Plaza Bucharest Convention Center.

```
(instance |Hotel2517| |Hotel|)
(attribute-filler |Hotel2517| "HotelHotel␣Zimbru" name)
(instance |CaleaBucovinei 1-3, CampulungMoldovenesc, RO| |Address|)
(related |Hotel2517| |CaleaBucovinei 1-3, CampulungMoldovenesc, RO|
                                                        has-address)
(related |Hotel2517| |+40230314356| has-phone)
(related |Hotel2517| |www.suceava360.ro/| has-website)
```

Figure 11.8: The second ABox for asserting information about 2517 hotels in Romania.

```
(implies agriTourism (and tourism (some based agriculture)))
(implies nauticalTourism (and tourism (some based sailingAndBoating)))
(implies medicalTourism (and tourism (some based medicalTreatment)))
(implies culinaryTourism (and tourism (some based localFoodTasting)))
(implies popCultureTourism (and tourism (some based entertainment)))
(implies culturalTourism (and tourism (some based regionCulture)))
(implies extremeTourism (and tourism
                            (some based visitingDangerousPlaces)))
(implies sexTourism (and tourism (some based sexualActivity)))
(implies geotourismTourism (and tourism (some based
                                geologicalAndGeomorphologically)))
(implies heritageTourism (and tourism (some based culturalHeritage)))
(implies warTourism (and tourism (some based warZones)))
(implies lgbtTourism (and tourism (some based sexualOrientation)))
(implies wellnessTourism (and tourism (some based healthAndWellBeing)))
(implies wildlifeTourism (and tourism
                              (some based animalsInNaturalHabitat)))
```

Figure 11.9: Modelling various types of tourism.

```
(implies Basketball activity)        (implies BeachVolleyball activity)
(implies Biking activity)            (implies BootCamp activity)
(implies Calisthenics activity)      (implies ChoppingFirewood activity)
(implies CircuitTraining activity)   (implies DanceRevolution activity)
(implies EllipticalTrainer activity) (implies FamilyHike activity)
(implies FieldHockey activity)       (implies FlagFootball activity)
(implies Football activity)          (implies HalfMarathon activity)
(implies Handball activity)          (implies Hockey activity)
(implies IceHockey activity)         (implies IceSkating activity)
(implies InlineSkating activity)     (implies IrishDancing activity)
```

Figure 11.10: Touristic activities.

```
(related IoanaPension Basketball hasActivity)
(related IoanaPension Biking hasActivity)
(related IoanaPension ChoppingStackingFirewood hasActivity)
(related IoanaPension FamilyHike hasActivity)
(related IoanaPension PingPong hasActivity)
```

Figure 11.11: Activities around accomodation places.

```
(instance |Padurea Clujului| |Forest|)
(instance |Oaza Acvatica relicta in Campia Romana| |Oasis|)
(instance |Muntii Locvei| |Mountain|)
(instance |Muntii Lotrului| |Mountain|)
(instance |Muntii Macin| |Mountain|)
(instance |Pasul Prislop| |Pass|)
(instance |Pasul Paltinis| |Pass|)
(instance |Platoul Bucegi| |Plateau|)
(instance |Golful Musura| |Bay|)
(instance |Golful Holobina| |Bay|)
(instance |Canalul Dunarii| |Canal|)
(instance |Canalul Timis| |Canal|)
(instance |Chetarul Focul Viu| |Ice|)
(instance |Cascada Rachitele| |Waterfall|)
```

Figure 11.12: Geographical objectives.

```
(implies |Club| |Entertainment|)
(implies |Racetrack| |RecreationalStructure|)
(implies |Country| |GeopoliticalEntity|)
(implies |accommodation| |PublicPlace|)
(implies |Factory| |BusinessPlace|)
(implies |City| |GeopoliticalEntity|)
(implies |Waterfall| |WaterFeature|)
(implies |Beach| |GeographicalFeature|)
(implies |TravelPoint| |TransportationStructure|)
(implies |HistoricalSite| |TouristStructure|)
(implies |CommunityCenter| |GovernmentStructure|)
(implies |PoliceStation| |GovernmentStructure|)
(implies |JazzClub| |Club|)
(implies |Pass| |GeographicalFeature|)
(implies |BusinessPlace| |Place|)
(implies |Mountain| |GeographicalFeature|)
(implies |Place| |SpatialEntity|)
(implies |FoodStore| |ShoppingFacility|)
(implies |ConventionCenter| |BusinessPlace|)
(implies |HigherEducationFacility| |EducationalStructure|)
(implies |Market| |ShoppingFacility|)
(implies |MedicalStructure| |PublicPlace|)
(implies |CityHall| |GovernmentStructure|)
(implies |EatingAndDrinking| |PublicPlace|)
```

Figure 11.13: Geographical features.

```
(related Romania |NORD-EST| has-region)
(related Romania |SUD-EST| has-region)
(related Romania |SUD-MUNTENIA| has-region)
(related Romania |SUD-VEST-MUNTENIA| has-region)
(related Romania |VEST| has-region)
(related Romania |NORD-VEST| has-region)
(related Romania |CENTRU| has-region)
(related Romania |BUCURESTI-ILFOV| has-region)

(related Romania Transilvania has-subdivision)
(related Romania Moldova has-subdivision)
(related Romania Tara$\_$Romaneasca has-subdivision)

(related |SUD-EST| |CONSTANTA| has-county)
(related |CENTRU| |COVASNA| has-county)

(related |ALBA| |Comuna Albac| has-commune)
(related |ALBA| |Comuna Almasu Mare| has-commune)
(related |ALBA| |Comuna Arieseni| has-commune)

(instance |Comuna Albac| |Commune|)
(instance |Comuna Almasu Mare| |Commune|)
(instance |Comuna Arieseni| |Commune|)
```

Figure 11.14: Sample of information for administrative regions.

```
(instance |Muzeul "Mitropolit?␣Antim␣Ivireanul"| |Museum|)
(instance |Muzeul Memorial "C.I.␣si␣C.C.␣Nottara"| |Museum|)
(instance |Casa Memoriala "Constantin␣Joja"| |Museum|)
(instance |Muzeul de Arta Populara "Prof.Dr.Nicolae␣Minovici"| |Museum|)
(instance |Muzeul Memorial "Dr.␣Victor␣Babes"| |Museum|)
(instance |Muzeul de Arta Veche "ing.␣Dumitru␣Minovici"| |Museum|)
(instance |Colectia de Arta Plastica "Cecilia␣Cutescu␣Storck"| |Museum|)
(instance |Casa Memoriala "George␣si␣Agatha␣Bacovia"| |Museum|)
(instance |Muzeul Memorial "George␣Calinescu"| |Museum|)
(instance |Muzeul National "George␣Enescu"| |Museum|)
```

Figure 11.15: Sample from the 637 Romanian museums.

```
(related |Muzeul "Mitropolit␣Antim␣Ivireanul"|
         |Str. Antim nr. 29, sector 5| |hasLocation|)
(related |Muzeul de Arta Populara "Prof.␣Dr.␣Nicolae␣Minovici"|
\hspace{4cm} |muzeul.minovici@muzeulbucurestiului.ro| has-email)
(related |Muzeul Memorial "C.C.␣Nottara"| |1956| has-founding-year)
(related |Muzeul "Mitropolit␣Antim␣Ivireanul"| |44,426141| has-latitude)
(related |Muzeul "Mitropolit␣Antim␣Ivireanul"| |26,093867| has-longitude)
(related |Bucuresti| |Muzeul "Mitropolit␣Antim␣Ivireanul"|
         county-has-museum)
(related |Muzeul "Mitropolit␣Antim␣Ivireanul"| |614.10.60| has-phone)
(related |Muzeul Memorial "C.I.␣si␣C.C.␣Nottara"|
         |9:00 - 16.30; luni: inchis| has-schedule)
(related |Muzeul de Arta Populara "Prof.␣Dr.␣Nicolae␣Minovici"|
         |http://www.minovici.ro/| has-website)
```

Figure 11.16: Features for the 737 museums.

```
(instance |Muzeul "Mitropolit? Antim Ivireanul"| |Museum| )
(instance |Metropolitan Bishop Antim Ivireanu Museum| |Museum|)
(same-as |Metropolitan Bishop Antim Ivireanu Museum|
         |Muzeul "Mitropolit Antim Ivireanul"|)
(same-as |Muzeul Memorial "C.I. si C.C. Nottara"|
         |C.I. and C.C. Nottara Memorial Museum|)
```

Figure 11.17: Correspondence between Romanian and English museum names.

```
(instance |Pestera Zamonita| |Cave|)
(instance |Pestera Grota Zanelor| |Cave|)
(instance |Pestera Zanelor| |Cave|)
(instance |Pestera de la Zapodie| |Cave|)
(instance |Pestera Zaton| |Cave|)
(instance |Avenul cu Zapada din Scorota Seaca| |Cave|)
(instance |Avenul Zbegului| |Cave|)
(instance |Pestera Zeicului| |Cave|)
(instance |Pestera I de la Avenul de sub Zgurasti| |Cave|)
(instance |Pestera de la Zvarlus| |Cave|)
(instance |Pestera de la Zvarlusul Corbestilor| |Cave|)
(instance |Pestera de la Zvarlusul soimului| |Cave|)
```

Figure 11.18: Sample from the 953 caves in Romania.

```
(instance |Muntii Bihorului| |Mountain|)
(instance |Muntii Capatanii| |Mountain|)
(instance |Muntii Aninei| |Mountain|)
(instance |Muntii Padurea Craiului| |Mountain|)
(instance |Muntii Sebes| |Mountain|)

(related |ALBA| |Muntii Bihorului| has-mountain)
(related |CARAS-SEVERIN| |Muntii Aninei| has-mountain)
(related |CARAS-SEVERIN| |Culmea Mehadiei| has-mountain)
```

Figure 11.19: Romanian mountains in the tourism ontology.

```
(instance |Portul Braila| |Port|)
(instance |Portul Constanta| |Port|)
(instance |Aeroportul International Arad| |Airport|)
(instance |Aeroportul International "George␣Enescu" Bacau| |Airport|)
(instance |A1-Bucuresti-Nadlac| |Highway|)
(instance |A2-Autostrada Soarelui| |Highway|)
(instance |Gradina Zoo Barlad| |Zoo|)
(instance |Gradina Zoo Bacau| |Zoo|)
(instance |Centrul nr1 Cluj Napoca| |InformationCenter|)
(instance |Centrul nr1 Bistrita| |InformationCenter|)
(instance |Situl arheologic de la Callatis| |HistoricalSite|)
(instance |Situl arheologic de la Histria| |HistoricalSite|)
(instance |AFI Palace Cotroceni Shopping center| |ShoppingCenter|)
(instance |Baneasa Shopping City Shopping center| |ShoppingCenter|)
(instance |Piata Unirii din Timisoara| |Market|)
(instance |Piata Mare din Sibiu| |Market|)
(instance |Templul Mare din Radauti| |Temple|)
(instance |Sinagoga veche din Fabric sec. XVIII| |Synagogue|)
(instance |Moscheea Noua Constanta Litoralul Romanesc| |Mosque|)
(instance |Catedrala Acoperamantul Maicii Domnului| |Cathedral|)
(instance |Manastirea Voronet Capela sixtina a estului| |Chapel|)
(instance |Partia Vartop Ariseni| |SkiArea|)
(instance |PARCUL SZEKELY LASZLO| |PlayingField|)
(instance |Parcul Calimani| |Park|)
(instance |Baza Sportiva Record Cluj-Napoca| |Gym|)
(instance |Selas Golf Polo Club| |GolfCourse|)
(instance |Spitalul Clinic Judetean de Urgenta Cluj| |Hospital|)
(instance |Penitenciarul Aiud Alba| |Prison|)
(instance |Directia Politiei Rutiere| |PoliceStation|)
(instance |ALBA Biblioteca LUCIAN BLAGA| |Library|)
(instance |Curtea de Apel Alba Iulia| |Courthouse|)
(instance |Teatrul "Maria␣Filotti" Braila| |Theater|)
(instance |Stadionul Unirea Ungheni Mures| |Stadium|)
(instance |Satul de Vacanta| |AmusementPark|)
(instance |Gradina Botanica "Alexandru␣Borza" a Universitatii
\hspace*{4cm}Babes-Bolyai din Cluj-Napoca| |Garden|)
(instance |Sigma bussines center| |BusinessCenter|)
(instance |Ramada Plaza Bucharest Convention Center| |ConventionCenter|)
```

Figure 11.20: Sample from the 729 additionally POI instances.

```
public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {
 writer = new PrintWriter("output.racer", "US-ASCII");
 individualCounter = 0;
 writer.println("(full-reset)");
 writer.println("(define-concrete-domain-attribute hasLati :domain geolocationCoordinate :type real)");
 writer.println("(define-concrete-domain-attribute hasLongi :domain geolocationCoordinate :type real)");
 writer.println("(define-primitive-role hasGeolocationCoordinate :domain node :range geolocationCoordinate)");
 writer.println("(define-primitive-role hasNode :domain road :range node)");
```

Figure 11.21: Generating TBox in KRSS syntax for the Open Street Map vocabulary.

(Fig. 11.5). The JAVA code uses the OSMOSIS API[1]

Inporting individuals from Foursquare

The main use case of the ontology is to create a guide application for tourists coming to the city of Cluj-Napoca.

The Foursquare taxonomy and individuals can be imported in the ontology, as follows:

```
(implies Restaurant EatingAndDrinking)
(implies Afghan_Restaurant Restaurant)
(implies African_Restaurant Restaurant)
(implies American_Restaurant Restaurant)
(implies Arepa_Restaurant Restaurant)
(implies Argentinian_Restaurant Restaurant)
```

```
(implies Aquarium ArtsEntertainment)
(implies Arcade ArtsEntertainment)
(implies Art_Gallery ArtsEntertainment)
(implies Bowling_Alley ArtsEntertainment)
(implies Casino ArtsEntertainment)
(implies Comedy_Club ArtsEntertainment)
(implies Concert_Hall ArtsEntertainment)
```

```
(implies Animal_Shelter ProfessionalOtherPlaces)
(implies Auditorium ProfessionalOtherPlaces)
(implies Building ProfessionalOtherPlaces)
(implies Convention_Center ProfessionalOtherPlaces)
(implies Meeting_Room ProfessionalOtherPlaces)
(implies Event_Space ProfessionalOtherPlaces)
```

```
(implies Bar NightLifeSpot)
(implies Beer_Garden NightLifeSpot)
(implies Cocktail_Bar NightLifeSpot)
(implies Dive_Bar NightLifeSpot)
```

```
(implies Athletics_&_Sports OutdoorsRecreation)
(implies Baseball_Field OutdoorsRecreation)
(implies Basketball_Court OutdoorsRecreation)
(implies Golf_Course OutdoorsRecreation)
(implies Hockey_Field OutdoorsRecreation)
```

[1]http://wiki.openstreetmap.org/wiki/Osmosis#Downloading.

Figure 11.22: Architecture for the Lela Foursquare module.

Table 11.1: Linking available datasets.

Data set	**Available at**	**Description**
Foursquare	https://foursquare.com/	Location-based social networking
Romania Museum Guides	http://data.gov.ro/dataset	Descriptive data and geolocations of 967 museums in Romania
Wikipedia	http://wikipedia.ro	Various categories about Romanian touristic places
Freebase	http://www.freebase.com/	Community-curated database of well-known people, places and things - some about Romania
Geonames	http://www.geonames.org/	Covers all countries and contains over eight million place names - some about Romania
Wikisherpa	http://www.wikisherpa.com/	Data from wikiTravel in a more structured way
DBpedia	http://dbpedia.org/	Structured data from wiki to other external resources
Cluj4All	cluj4all.com	Around 7000 objectives about Cluj-Napoca

```
1 (implies Antique_Shop ShopService)
2 (implies Arts_&_Crafts_Store ShopService)
3 (implies Automotive_Shop ShopService)
4 (implies Bank ShopService)
5 (implies Bike_Shop ShopService)
```

To develop the ontology, we followed the methodology in [36] and we also enact various ontology design patterns [44].

Data collection and fusion

We focused on discovering and selecting the relevant open data sets for the Romanian tourism domain. Touristic points of interest are collected from tow sources: i) the available *POI data* sources (Wikipedia, Freebase, DBPedia, Geonames, Wikisherpa, Wikitravel) and ii) Named Entity recognition from touristic blogs. Data fusion is performed in AllegroGraph and saved as a triple store[2], while RacerPro server is used for reasoning on the Lela ontology.

The following sources were exploited: *Wikipedia*, *DBpedia*, *Geonames*, *Freebase*, *Wikisherpa*, *WikiTravel* (table 11.1). The data sets provided by various Romanian governmental agencies were used containing information for Romanian museums, churches, and historical points.

Events. The concept *Event* is an *Activity* and has as direct subconcepts *Exposition* or *Concert*.

[2]Available at http://www.recognos.ro/lela/LelaLinkedDataSet.nq

```
? (describe-concept Event)
> (Event
        :told-primitive-definition Activity
        :synonyms (Event)
        :parents ((Activity))
        :children ((Exposition) (Concert)))
```

Listing 11.1: Modelling knowledge abount events.

```
(implies Event (exactly 1 hasEventContent))
(implies Event (all hasPeriod (or DatePeriod DateTimePeriod)))
(implies Event (exactly 1 isInSite))
(implies Event (all hasPrice TicketPrice))
(implies Event (some hasPeriod (or DatePeriod DateTimePeriod)))
(implies TimePeriod Period)

(implies Exhibition EventContent)
(implies Movie EventContent)
(implies Show EventContent)
(implies Party EventContent)
(implies Competition EventContent)
(implies Concert EventContent)

(implies DatePeriod Period)
(implies Month Period)
(implies Weekday Period)
(implies Season Period)
(define-concept Month (or January February March April May June July
                          August September October November December))
(define-concept Weekday (or
            Monday Tuesday Wednesday Thursday Friday Saturday Sunday))
```

Price. Some knowledge about price is also required inthe tourism domain. Price-related concept are described in listing 11.2.

Listing 11.2: Introducing price-related concepts.

```
(implies Price (exactly 1 hasCurrency))
(implies TicketPrice Price)
(implies GastroPrice Price)
(implies RoomPrice (and Price (at-least 1 hasRoom)))
(implies DrinkPrice GastroPrice)
(implies FoodPrice GastroPrice)

(define-primitive-role hasBreakfastPrice :parent hasPrice)
(define-primitive-role hasparkingPrice :parent hasPrice)
(define-primitive-role hasRoomPrice :parent hasPrice)
(define-primitive-role hasFee :parent hasPrice)

(implies Site (all hasOpeningTime DateTimePeriod))
(implies PaidAttraction POI)
(implies PaidAttraction (all hasPrice TicketPrice))
(implies PaidAttraction (some hasPrice TicketPrice))

(implies NotPaidAttraction POI)
(disjoint PaidAttraction NotPaidAttraction)
```

Infrastructure. In some cases, the access infrastructure to some touristic objectives (see listing 11.3).

Listing 11.3: Formalising the infrastructure of a location.

```
(implies AdministrationInfrastructure Infrastructure)
(implies PublicInfrastructure Infrastructure)
(implies TerminalInfrastructure Infrastructure)
(implies ShoppingInfrastructure Infrastructure)
(implies HealthInfrastructure Infrastructure)
(implies SportsInfrastructure Infrastructure)
(implies EntertainmentInfrastructure Infrastructure)
(implies EntertainmentInfrastructure (all hasPrice TicketPrice))

(implies CarParking PublicInfrastructure)
(implies Bank PublicInfrastructure)
(implies GasStation PublicInfrastructure)
(implies PostOffice PublicInfrastructure)
(implies Library PublicInfrastructure)

(implies MetroStation TerminalInfrastructure)
(implies TrainStation TerminalInfrastructure)
(implies BusStation TerminalInfrastructure)
(implies Airport TerminalInfrastructure)

(implies Theatre EntertainmentInfrastructure)
(implies Cinema EntertainmentInfrastructure)

(implies Gym SportsInfrastructure)
(implies Stadium SportsInfrastructure)

(implies Clinic HealthInfrastructure)
(implies Pharmacy HealthInfrastructure)
(implies Hospital HealthInfrastructure)

(implies ConferenceRoomPrice (all hasRoom ConferenceRoom))
(implies ConferenceRoomPrice RoomPrice)
(implies GuestRoomFacility RoomFacility)
```

Assume that the blog 101 states that "at NapoliCentrale one can eat well": The informatios in structured as follows:

```
(define-primitive-role hasCuisineQuality :domain Gastro :range Match)

(instance m12345 Match)
(instance b101 BlogPost)
(related m12345 b101 fromBlogPost)
(attribute-filler m12345 "la␣pNC␣se␣mananca␣bine" hasText)
(attribute-filler m12345 1.0 hasScore)
(related pizzeriaNapoliCentrale m12345 hasCuisineQuality)
```

Retrieving information from the ontology

The following listing illustrates three operations: i) checking the ontology consistency, ii) retrieve information about individuals in the ontology andrea iii) identify the sub-concepts of the main five axis of the ontology

```
(tbox-cyclic?)
```

```
(tbox-coherent?)

(describe-individual m12345)
(describe-individual m67890)

(concept-instances Location)
(concept-instances Activity)
(concept-instances POI)
(concept-instances Gastro)
(concept-instances Accommodation)
```

To obtain all activities and their locations, one can use the following nRQL query:

```
(retrieve (?activity ?location) (and (?activity Activity)
                                     (?activity ?location hasLocation)
))
```

To obtain all eating and drinking options and their locations, the following query can be enacted:

```
(retrieve (?ed) (and (?ed Gastro)
                     (?ed ?location hasLocation)
))
```

For retrieving all instances which are not Accommodation:

```
(retrieve (?x) (?x (not Accommodation)))
```

To list all touristic objective with a positive review score greater than 0.8, we can use the RacerPro command:

```
(concept-instances (and Match (>= hasScore 0.8)))
```

The corresponding nRQL query is:

```
(retrieve (?x (hasScore ?x) (told-value (hasScore ?x)))
(?x (and Match (an hasScore) (> hasScore 0.8))))
```

To enumerate all points of interest and activities for which the location is explicitly specified we can use:

```
(retrieve (?x) (?x (has-known-successor POI_has_Location)))
(retrieve (?x) (?x  (has-known-successor Activity_has_Location)))
```

Chapter 12

Engineering a Vehicular Network Ontology

To develop the Vehicular Network (VANET) ontology, we follow the methodology in [36] and we also enact various ontology design patterns [44]. The engineering steps presented in this chapter are [21]: i) defining competency questions, ii) re-using other ontologies, iii) defining main concepts and roles, iv) populating the ontology, and v) ontology debugging and evaluation.

12.1 Competency questions

The competency questions (CQs) help to define the limits of the domain to be modeled and also to identify the main concepts and roles of the ontology. Examples of CQs for VANET domain are listed in Table 12.1. These CQs help to identify the concepts (i.e. `Lane`, `Speed`, `Overtake`, `MultiHop`, `EmergencyVehcile`) and roles (i.e `isLocated`, `hasSpeed`, `nearby`).

Table 12.1: Sample of competency questions for the vehicular network ontology.

No	*Competency question*
CQ_1	Which are the vehicles on the same lane within a specific area?
CQ_2	Which data is available about the closest vehicle in front/behind?
CQ_3	Which is the closest vehicle approaching from opposite direction?
CQ_4	Which is the average speed for the next 5km?
CQ_5	Is it safe to change lane?
CQ_6	Is it safe to overtake the vehicle in front?
CQ_7	Which vehicles in the VANET can perform multi-hop routing?
CQ_8	Are there any emergency vehicles in the nearby?

12.2 Reusing other ontologies

Two categories of ontologies can be reused: vehicular related domain ontologies and general ontologies. The general knowledge required to model various facets of vanets include: spatial, temporal, situation awareness.

Several related ontologies in the automotive domain have been developed [13, 43, 3, 37, 14]. They have focused on modeling specific facets of a vehicular network, but no one had the goal to cover the entire vanets domain. However, part of the following related ontologies were considered for re-used during the engineering of our vehicular network ontology.

An ontology of vehicular networks security has been modeled in [13]. The aim has been to classify the vulnerabilities based on the impact of the intrusion and functionality affected in routing protocols. The top level concepts are *Attack*, *Consequences*, *Vulnerabilities*, and *Actor*. The attacks are either active (like relaying attacks, injection of malicious code, modifying of packets) or passive (sybil, illegal eavesdropping). The identified vulnerabilities are: high mobility, the cooperative relationships, shared wireless medium. Among the consequences modeled by the ontology, there are: degradation of vehicular network performance, road congestion, insulation of nodes or even road accidents.

The ontology in [43] aims to determine the autonomy layer of an automated vehicle. The main use case is at self-assessment of the perception system to monitor co-driving. The module designed for situation assessment formalises knowledge such as: environment conditions, moving obstacles, driver state, navigable space, which are also relevant concepts for the vehicular network domain.

The CAOVA (Car Accident lightweight Ontology for VANETs) structures information from two sources: i) collected from vehicle sensors when an accident occurs, or ii) imported from the the General Estimates System accidents database [3]. Due to the private character of data formalised for the driver concept (blood, allergies, illness, medication, pregnant, weight, age, sex), the ontology has been encrypted using the Advanced Encryption Standard. The top level concepts are: *Vehicle*, *Accident*, *Environment*, and *Occupant*. The use cases of the ontology regard various car safety application, with the goal to increase the interoperability among emergency services, authorities, or other vehicles.

Having the main objective to facilitate vehicle selling, several automotive ontologies have been designed to be used in combination with the GoodRelations [28] commercial oriented vocabulary. In this line, from the Volkswagen Vehicles Ontology[1] or Vehicle Sales Ontology[2] some concepts are also relevant in the context of vehicular communication: model, dimensions of the vehicle, engine, type of the vehicle (such as van, truck, etc.).

A vehicular ontology is proposed in [37] having the role of a dynamic middleware between two vanets. The ontology focuses on defining packets and their features (MFR-BroadcastPacket, PositionBasedPacket, ClusterBasedPacket, etc). The axioms of these packets are used to infer the meaning of a packet and to classify it under the most appropriate routing strategy. The work in [37] can be integrated in the larger context of semantic middleware solutions, as it employs ontologies to achieve communication protocol interoperability.

Aiming to facilitate the identification of reusable knowledge, we designed the ontology to contain several modules. An ontology module represents a shared, domain-independent conceptualization intended to be used for different tasks and applications. From this perspective, ontology modules are comparable to software libraries in the software engineering domain. In this way, different traffic scenarios can enact only knowledge relevant for the application domain.

[1] http://www.volkswagen.co.uk/vocabularies/vvo/ns

[2] http://www.heppnetz.de/ontologies/vso/ns

```
21. (in-tbox Communication)
22. (implies (or SafetyApplication Infotainment ResourceEfficiency) Application)
23. (implies (or Warning PassiveSafety ActiveSafety ProActiveSafety) SafetyApplication)
24. (implies (or QuickWarningAlerts NormalWarningAlerts) Warning)
25. (implies CollisionAvoidance ProActiveSafety)
26. (implies LaneChanging ProActiveSafety)
27. (implies Overtaking (and LaneChanging CollisionAvoidance))
28. (implies NormalTrafficAlerts ResourceEfficiency)
29. (implies AutonomousSystems ResourceEfficiency)
30. (implies GreenLightWave NormalTrafficAlerts)
31. (implies EnhancedRouteGuidance NormalTrafficAlerts)
32. (implies CooperativPlatooning AutonomousSystems)
33. (implies AdHocServices Infotainment)
```

Figure 12.1: Top level taxonomy of vehicular applications.

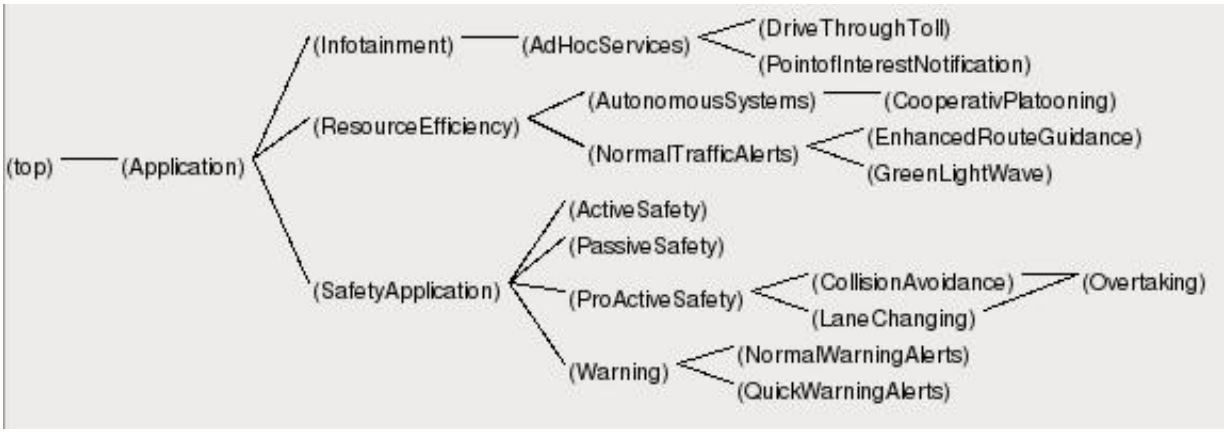

Figure 12.2: View on the taxonomy of vehicular applications.

12.3 Defining main concepts and roles.

The main elements of the ontology are organised on modules like: communication, vehicular, traffic hazards, etc, as follows:

The *communication module* defines basic communication patterns and messages that take place in vanets-enabled applications. We start by classifying the main application types (Fig. 12.1) by enacting the taxonomy ontology design pattern [44]. The applications of vehicular communication [45] can be split into three subcategories: safety, resource efficiency and infotainment (axiom 22). Note for instance that, the overtaking maneuver implies both lane changing and collision avoidance safety issues (line 27).

The communication regimes are classified according to the transmission scheme into bidirectional and position based (line 34 in Fig. 12.3). The bidirectional regime (or unicast) enables connection between two vehicles by performing four phases: discovery, connection, data, and ending (lines 35-37). The position based regime (or geocast) in lines 38-39 simultaneously conveys information one way to a group of vehicles in a specified geographical area.

After discovering the vehicles in the area of interest, the information tagged with the geographical area is sent in the flooding phase. The acknowledgment is skipped in the position based regime (line 40), where `bottom` is the empty concept, while the control channel is eliminated in the fast bidirectional sub-regime (axiom 42). With axioms 43-44,

```
34. (implies CommunicationRegimes (or Bidirectional PositionBased))
35. (implies Bidirectional (and (=1 hasTarget (or Vehicle RoadSideUnit))
36.                             (some hasPhase Discovery)
                                (some hasPhase Connection)
37.                             (some hasPhase Data)
                                (some hasPhase  Ending)))
38. (implies PositionBased (and OneWay
                                (some hasTarget VehicleGroup)
39.                             (some hasPhase Discovery)
                                (some hasPhase Flooding)
40.                             (some hasAcknowledgement bottom)))
41. (equiv VehicleGroup (and (> 2 hasVehicle Vehicle)
                             (all hasArea GeoRegion)))
42. (implies FastBidirectional (and Bidirectional
                                    (some hasControlChannel bottom)))
43. (implies SingleHop PositionBased)
44. (implies MultiHop PositionBased)
```

Figure 12.3: Communication regimes.

both the single hop and multi hop protocols are classified as position based. If the area is large, the multi-hop routing mechanism is activated when the information needs to travel form one vehicle to another to reach all the targeted vehicles.

Different types are messages (alerts, beacons, normal messages) are conveyed in vanets applications (line 51 in Fig. 12.4). Permanent beacons and alert messages are sent using the position based communication regime (axiom 52). In line 53, the messages have a priority between 0 (highest) and 4 (lowest). The messages are also characterized by a specific update rate which leads to a reception probability known as *packet delivery ratio* (`PDR`).

For safety applications a minimum value of 0.95 is required for the `PDR` parameter (axiom 54). To guarantee the `PDR` value, multiple messages should be sent within the so-called time-to-live (`TTL`) of the message. However, in some safety applications, the message should be sent in real time (`RT` in line 55). Also relevant, the latency parameter represents the time delay from sending and receiving a packet. For instance, lane changing application requires a latency below 100ms (line 56).

Two disjoint transmission types exist in vehicular communication: vehicle-to-vehicle (`V2V`) or vehicle-to-infrastructure (`V2I`) (axioms 57-58). Different sub-types of `V2V` transmission can be envisaged: train to vehicle (`T2V`), drone to vehicle (`D2V`), vehicle-to bike (`V2B` in line 59). Road side units (`RSU`) represent the main infrastructure available for `V2I` (definition 60).

Normal warning alerts signal an event by sending multi-hop messages in a time window to other vehicles (Fig. 12.5). For instance, the rail collision warning is sent by the rail to the vehicles nearby, when the train is approaching a level-crossing area. Hazardous notification sends warning about possible hazards detected by vehicle sensors, road side units or driver. The *hazard module* in our ontology was designed to support these warning messages, as axiom 72 bears out. For instance, the ESP sensor of the vehicle may detect a slippery location or a pit on the roadway and warn other drivers behind about these possible dangerous situations. In our approach, the identified slippery locations represent specific instances that populate the ontology.

Populating the ontology. Fig. 12.6 illustrates an instantiation of the warning message

```
51. (implies (or Alert Beacons Normal) MessageType)
52. (implies Beacons (some hasCommunicationRegime PermanentBased))
53. (equiv Priority (one-of 0 1 2 3 4))
54. (implies SafetyApplication (> PDR 0.95))
55. (implies (or TTL RT) TimeCritical)
56. (implies LaneChanging (< Latency 100))
57. (implies (or V2V V2I) TransmissionType)
58. (disjoint V2V V2I)
59. (implies (or T2V D2V V2B) V2V)
60. (implies V2RSU V2I)
```

Figure 12.4: Message features in vanets communication.

```
61. (equiv NormalWarningAlerts (and Alert
62.           (some hasCommunicationRegime MultiHopPositionBased)
63.           (some hasApplicationType Warning)
64.           (some hasTransmissionType (or V2V V2RSU))))
65. (implies RailCollisionWarning NormalWarningAllerts)
66. (implies SlowVehicleWarning NormalWarningAllerts)
67. (implies LimitedAccessWarning NormalWarningAllerts)
68. (implies WorkingAreaWarning NormalWarningAllerts)
69. (implies PostCrashWarning NormalWarningAllerts)
70. (implies HazardousLocationNotification NormalWarningAllerts)
71. (implies TrafficJamAheadWarning NormalWarningAllerts)
72. (implies (or Pit SlipperyRoadWay WaterOnLane OilOnLane) Hazard)
```

Figure 12.5: Classifying warning alerts in vanets.

```
73. (in-abox hazard-location-notification)
74. (instance hln (and HazardousLocationNotification (= atPos p1))
75. (instance p1 (and Position (= hasLat 49.19205) (= hasLong 16.6131)))
76. (instance law (and LimitedAccesWarning (< height 2.4) (= atPos p1))
```

Figure 12.6: Assertions in the vanet ontology.

`HazardousLocationNotification`. The individual `hln` is an instance of the concept `HazardousLocationNotification` and it also specifies the position `p1` where the hazard was identified (line 74). The individual `p1` is of type `Position` constrained by two feature roles `hasLat` and `hasLong`. The `LimitedAccesWarning law` signals other vehicles a limitation of 2.4 meter height at the same position `p1`.

12.4 Ontology debugging and evaluation

. From the semantic point of view, the ontology was checked against consistency and coherence. After iterative repair steps, the ontology is consistent and the cycles in the concepts removed. The domain coverage was checked by validating the ontology against the CQs defined initially (recall table 12.1), thus assuring that the ontology is able to provide answers to the CQs specified.

Graph based evaluation metrics were analysed in order to avoid structured defects such as unbalanced subsumption branches, lazy or uninstantiated concepts, concepts with only one sub-concept, concepts with more than 25 sub-concepts and other elements considered worst practice in ontology engineering [50]. From the engineering perspective, the ontology uses several ontology design patterns: n-ary relationship, partition, universal-existential macro, modular design pattern, taxonomy, agent role, etc. [44].

Vanet-related application heavily rely on temporal and spatial reasoning. We exploit the RacerPro [23] capabilities for temporal reasoning and geospatial reasoning to facilitate semantic-based real time intelligent decisions.

12.5 Event recognition

Lane change assistance scenario. The requirements for lane changing application [45] are formalised in our ontology by enacting the *n-ary relations* ontology design pattern [44]. The aim of the pattern is to model n-ary relationship in an ontology, given that description logic has been designed to express binary relations only.

In Fig. 12.7 the `LaneChanging` is a `ProActiveSafety` application. The lane changing message is not targeted to a specific vehicle, but the message is beaconed to all vehicles nearby (lines 81-82). The `V2V` transmission type is needed to avoid sending messages to road side units. The `SingleHop` communication regime suffices (line 84), due to the current specifications of the IEEE 802.11p standard [29]. The `RealTime` constraint is introduced in line 85 to signal that the message is valid only for the current instance of time. In line 86, the highest priority `0` is used, a `PDR` value above 95%, with a latency of maximum `100ms` [45].

Detecting inconsistencies. The reasoning services of description logic can be used to validate different messages at the application level. For instance, a specific message sent through the `MultiHop` communication regime will not be classified as a `LangeChanging` message according to the definition in Fig. 12.7. Moreover, given the ABox:

```
ABox={  (instance ln LaneChanging),
        (related ln single hasCommRegime)}
```

the RacerPro signals an inconsistency relative to the TBox `lane-changing`. The query `(instantiators ln)` will print all the concepts to which the individual `ln` is an instance of.

```
81. (in-tbox lane-changing)
82. (equiv LaneChanging (and ProActiveSafety
                             (some hasMessageType Beacons)
83.                          (some hasTransmissionType V2V)
84.                          (some hasCommunicationRegime SingleHop)
85.                          (some hasTimeContraints RealTime)
86.                          (= hasPriority 0)
                             (> hasPDR 95)
                             (< hasLatency 100)))
```

Figure 12.7: Requirements for lane changing-related applications.

```
87. (instance c1 Skoda)
88. (instance c2 Dacia)
89. (define-event-assertion ((hasLocation c1 l1) 0 .1))
90. (define-event-assertion ((hasLocation c1 l2) .1 .2))
91. (define-event-assertion ((hasLocation c1 l3) .2 .3))
92. (define-event-assertion ((hasLocation c2 l1) .3 .4))
93. (define-event-assertion ((hasLocation c2 l2) .4 .5))
94. (instance l1 (and (= hasLat 49.19205) (= hasLong 16.6131)))
95. (instance l2 (and (= hasLat 49.19206) (= hasLong 16.6131)))
96. (instance l3 (and (= hasLat 49.19207) (= hasLong 16.6132)))
97. (equiv Lane1 (and (< hasLat 49.19206) (> hasLat 49.19204)
                     (= hasLong 16.6131)))
98. (equiv Lane2 (and (< hasLat 49.19206) (> hasLat 49.19204)
                     (= hasLong 16.6132)))
```

Figure 12.8: Geospatial and temporal reasoning.

```
101. (define-event-rule ((laneChange ?v ?l1 ?l2) ?t1 ?t2)
102.    ((?v vehicle) ?t0 ?tn)
103.    ((?l1 Lane1) ?t0 ?tn)
104.    ((?l2 Lane2) ?t0 ?tn)
105.    ((?v ?l1 hasLocation)) ?t0 ?t1)
106.    ((?v ?l2 hasLocation)) ?t2 ?t3))
```

Figure 12.9: Lane chaning event recognition.

Assertions about vehicles are valid only within a certain time interval. In Fig. 12.8, `c1` and `c2` are vehicles (axioms 87-88). Between time steps `[0.0, 0.1)ms` the individual `c1` is known to have location `l1`, between `[0.1,0,2)` location `l2`, and between `[0.2, 0.3)` location `l3` (lines 89-91). The locations are characterized by longitude and latitude coordinates, as transmitted through vehicular communication. From GIS maps, the definitions of `Lane1` and `Lane2` can be obtained (axioms 97-98). Based on these assertions, the system is able to deduce that locations `l1` and `l2` belong to the concept Lane1, while location `l3` to the concept `Lane2`, as the following RacerPro queries bear out:

```
1 ? (concept-instances Lane1)
2 > (l1, l2)
3 ? (concept-instances Lane2)
4 > (l3)
```

The event rule in Fig. 12.9 is used to recognise that an individual `?v` changes the lane, and also the instance of time when this event takes place. The rule signals that vehicle `?v` changes `Lane1` with `Line2` sometime between `?t1` and `?t2`. The variable `?v` is matched against objects of type `Vehicle` (line 102), `?l1` against locations that satisfy the definition of `Lane1` (line 103), respectively `?l2` against locations within the constraints of the concept `Lane2` (line 104). Consider that the vehicle `?v` was related to the location `?l1` via the role `has-location` within the time interval `[?t0, ?t1]` (line 105). The event is detected if the same vehicle `?v` appears in a different location `?l2` in a time interval starting with `?t2` (line 106).

The rule is fired in RacerPro engine and detects that `c1` has performed a lane changing maneuver, given the assertions in Fig. 12.8.

Appendix A

KRSS Syntax and Semantics

Syntax		Extension
Input	**Abstract**	
TOP	$\top$	$\Delta^{\mathcal{I}}$
BOTTOM	$\bot$	$\emptyset$
NUMBER		the numbers
INTEGER		the integers
STRING		the strings
(and C_1 ... C_n)	$C_1 \sqcap \cdots \sqcap C_n$	${C_1}^{\mathcal{I}} \cap \cdots \cap {C_n}^{\mathcal{I}}$
(or C_1 ... C_n)	$C_1 \sqcup \cdots \sqcup C_n$	${C_1}^{\mathcal{I}} \cup \cdots \cup {C_n}^{\mathcal{I}}$
(not C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
(all R C)	$\forall R{:}C$	$\{d \in \Delta_a^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$
(some R)	$\exists R$	$\{d \in \Delta_a^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \neq \emptyset\}$
(at-least n R)	$\geq n\, R$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d)\rvert \geq n\}$
(at-most n R)	$\leq n\, R$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d)\rvert \leq n\}$
(exactly n R)	$= n\, R$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d)\rvert = n\}$
(some R C)	$\exists R.C$	$\{d \in \Delta_a^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
(at-least n R C)	$\geq n\, R{:}C$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d) \cap C^{\mathcal{I}}\rvert \geq n\}$
(at-most n R C)	$\leq n\, R{:}C$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d) \cap C^{\mathcal{I}}\rvert \leq n\}$
(exactly n R C)	$= n\, R{:}C$	$\{d \in \Delta_a^{\mathcal{I}} \mid \lvert R^{\mathcal{I}}(d) \cap C^{\mathcal{I}}\rvert = n\}$

Table A.1: Concept Syntax and Semantics

Syntax		Extension
Input	**Abstract**	
top	$\top$	$\Delta_a^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
(and R_1 ... R_n)	$R_1 \sqcap \cdots \sqcap R_n$	${R_1}^{\mathcal{I}} \cap \cdots \cap {R_n}^{\mathcal{I}}$
(or R_1 ... R_n)	$R_1 \sqcup \cdots \sqcup R_n$	${R_1}^{\mathcal{I}} \cup \cdots \cup {R_n}^{\mathcal{I}}$
(not R)	$\neg R$	$(\Delta_a^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus R^{\mathcal{I}}$
(inverse R)	R^{-1}	$(R^{\mathcal{I}})^{-1} \cap (\Delta_a^{\mathcal{I}} \times \Delta^{\mathcal{I}})$
(range C)		$\Delta_a^{\mathcal{I}} \times C^{\mathcal{I}}$
(domain C)		$(C^{\mathcal{I}} \cap \Delta_a^{\mathcal{I}}) \times \Delta^{\mathcal{I}}$

Table A.2: Role Syntax and Semantics

Syntax		Semantics
Input	**Abstract**	
(and S_1 ... S_n)		${S_1}^{\mathcal{I}} \wedge \cdots \wedge {S_n}^{\mathcal{I}}$
(or S_1 ... S_n)		${S_1}^{\mathcal{I}} \vee \cdots \vee {S_n}^{\mathcal{I}}$
(not S)		$\neg S^{\mathcal{I}}$
(instance IN C)	$IN \in C$	$IN^{\mathcal{I}} \in C^{\mathcal{I}}$
(related IN I R)	$\langle IN, I \rangle \in R$	$\langle IN^{\mathcal{I}}, I^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
(equal IN_1 IN_2)		${IN_1}^{\mathcal{I}} = {IN_2}^{\mathcal{I}}$

Table A.3: Assertion Syntax and Semantics

Query	**Meaning**
(concept-subsumes? C_1 C_2)	$C_1 \Longrightarrow C_2$
(role-subsumes? R_1 R_2)	$R_1 \Longrightarrow R_2$
(individual-instance? IN C)	$IN \in C$
(individual-related? IN I R)	$\langle IN, I \rangle \in R$
(individual-equal? IN_1 IN_2)	$IN_1 = IN_2$
(individual-not-equal? IN_1 IN_2)	$\neg(IN_1 = IN_2)$

Table A.4: Query Syntax and Semantics

(concept-descendants C)
(concept-offspring C)
(concept-ancestors C)
(concept-parents C)
(concept-instances C)
(concept-direct-instances C)
(role-descendants R)
(role-offspring R)
(role-ancestors R)
(role-parents R)
(individual-types IN)
(individual-direct-types IN)
(individual-fillers IN R)

Table A.5: Retrieval Syntax

Bibliography

[1] Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking ontologies with AKTiveRank. In *Proc. of the International Semantic Web Conference, ISWC*, pages 5–9. Springer-Verlag, 2006.

[2] Franz Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[3] Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J Martinez, J-C Cano, Carlos T Calafate, and Pietro Manzoni. Caova: A car accident ontology for vanets. In *Wireless Communications and Networking Conference (WCNC)*, pages 1864–1869. IEEE, 2012.

[4] Stefano Borgo and Claudio Masolo. Foundational choices in DOLCE. In *Handbook on ontologies*, pages 361–381. Springer, 2009.

[5] John G Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. Sioc: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142, 2006.

[6] Dan Brickley and Libby Miller. FOAF vocabulary specification 0.98. *Namespace Document*, 9, 2012.

[7] Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual modeling of OWL DL ontologies using UML. In *The Semantic Web Conference*, pages 198–213. Springer, 2004.

[8] Jordi Conesa, Veda C Storey, and Vijayan Sugumaran. Usability of upper level ontologies: The case of researchcyc. *Data & Knowledge Engineering*, 69(4):343–356, 2010.

[9] Oscar Corcho, Catherine Roussey, LM Vilches-Blázquez, and Iván Perez Dominguez. Pattern-based OWL ontology debugging guidelines. CEUR Workshop Proceedings, 2009.

[10] Dublin Core Metadata Initiative et al. Dublin Core metadata element set, version 1.1. 2012.

[11] Mathieu d'Aquin and Enrico Motta. Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63, 2011.

[12] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[13] M Erritali, B El Ouahidi, B Hssina, B Boukhalene, and A Merbouha. An ontology-based intrusion detection for vehicular ad hoc networks. *Journal of Theoretical & Applied Information Technology*, 53(3):410–414, 2013.

[14] Michael Feld and Christian Müller. The automotive ontology: managing knowledge inside the vehicle and sharing it between cars. In *3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 79–86. ACM, 2011.

[15] Dimitrios Galanis, George Karakatsiotis, Gerasimos Lampouras, and Ion Androutsopoulos. An open-source natural language generator for OWL ontologies and its use in Protégé and Second Life. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*, pages 17–20. Association for Computational Linguistics, 2009.

[16] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. *Modelling ontology evaluation and validation*. Springer, 2006.

[17] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In *Knowledge engineering and knowledge management: Ontologies and the semantic Web*, pages 166–181. Springer, 2002.

[18] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on Ontologies*, pages 221–243. Springer, 2009.

[19] Edward F Gehringer and Carolyn S Miller. Student-generated active-learning exercises. In *ACM SIGCSE Bulletin*, volume 41, pages 81–85. ACM, 2009.

[20] Pierre Grenon and Barry Smith. Snap and span: Towards dynamic spatial ontology. *Spatial cognition and computation*, 4(1):69–104, 2004.

[21] A. Groza, A. Marginean, and V. Muresan. An ontology-based model for vehicular ad-hoc networks. In *18th IEEE International Conference on Intelligent Engineering Systems (INES2014), 3-5 July, Tihany, Hungary*. IEEE, 2014.

[22] Nicola Guarino. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, volume 46. IOS press, 1998.

[23] V Haarslev, R Möller, and M Wessel. RacerPro user's guide and reference manual version 2.0. *Racer Systems GmbH & Co. KG, editor. Germany: Hamburg*, 2012.

[24] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal*, 3(3):267–277, 2012.

[25] Jens Hartmann, Raúl Palma, and Asunción Gómez-Pérez. Ontology repositories. In *Handbook on Ontologies*, pages 551–571. Springer, 2009.

[26] Orit Hazzan, Tami Lapidot, and Noa Ragonis. Lab-based teaching. In *Guide to Teaching Computer Science*, pages 119–141. Springer, 2011.

[27] J Heinsohn, B Hollunder, J Muller, B Nebel, W Nutt, and HJ Profitlich. Terminological knowledge representation: A proposal for a terminological logic. 1990.

[28] Martin Hepp. Goodrelations: An ontology for describing products and services offers on the web. In *Know. Eng.: Practice and Patterns*, pages 329–346. Springer, 2008.

[29] Daniel Jiang and Luca Delgrossi. IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments. In *Vehicular Technology*, pages 2036–2040. IEEE, 2008.

[30] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods - a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007.

[31] Dionysios D Kehagias, Ioannis Papadimitriou, Joana Hois, Dimitrios Tzovaras, and John Bateman. A methodological approach for ontology evaluation and refinement. In *ASK-IT Final Conference. June.(Cit. on p.)*, 2008.

[32] Racer Systems GmbH & Co. KG. Racer reference manual version 1.9, 2005.

[33] Racer Systems GmbH & Co. KG. Racer user guide version 2.0, 2012.

[34] Jeffrey J McConnell. Active and cooperative learning: tips and tricks. *ACM SIGCSE Bulletin*, 37(2):27–30, 2005.

[35] Alistair Miles and José R Pérez-Agüera. Skos: Simple knowledge organisation for the web. *Cataloging & Classification Quarterly*, 43(3-4):69–83, 2007.

[36] Natalya F Noy, Deborah L McGuinness, et al. Ontology development: A guide to creating your first ontology, 2001.

[37] Vatsala Nundloll, Paul Grace, and Gordon S Blair. The role of ontologies in enabling dynamic interoperability. In *Distributed Applications and Interoperable Systems*, pages 179–193. Springer, 2011.

[38] Daniel Oberle, Stephan Grimm, and Steffen Staab. An ontology for software. In *Handbook on ontologies*, pages 383–402. Springer, 2009.

[39] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice. com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[40] Chintan Patel, Kaustubh Supekar, Yugyung Lee, and EK Park. OntoKhoj: a semantic web portal for ontology searching, ranking and classification. In *Proceedings of the 5th ACM international workshop on Web information and data management*, pages 58–61. ACM, 2003.

[41] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[42] Silvio Peroni, David Shotton, and Fabio Vitali. The Live OWL documentation environment: a tool for the automatic generation of ontology documentation. In *Knowledge Engineering and Knowledge Management*, pages 398–412. Springer, 2012.

[43] Evangeline Pollard, Philippe Morignot, and Fawzi Nashashibi. An ontology-based model to determine the automation level of an automated vehicle for co-driving. In *FUSION*, pages 596–603. IEEE, 2013.

[44] Jeffrey T Pollock and Ralph Hodgson. Ontology design patterns. *Adaptive Information: Improving Business through Semantic Interoperability, Grid Computing, and Enterprise Integration*, pages 145–194.

[45] Radu Popescu-Zeletin, Ilja Radusch, and Mihai Adrian Rigani. *Vehicular-2-X Communication: State-of-the-Art and Research in Vehicular Ad hoc Networks*. Springer, 2010.

[46] María Poveda, Mari del Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Common pitfalls in ontology development. In Pedro Meseguer, Lawrence Mandow, and Rafael M. Gasca, editors, *CAEPIA*, volume 5988 of *Lecture Notes in Computer Science*, pages 91–100. Springer, 2009.

[47] Valentina Presutti, Enrico Daga, Aldo Gangemi, and Eva Blomqvist. extreme design with content ontology design patterns. In *Proc. Workshop on Ontology Patterns, Washington, DC, USA*. Citeseer, 2009.

[48] Valentina Presutti and Aldo Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Conceptual Modeling-ER 2008*, pages 128–141. Springer, 2008.

[49] Mel Silberman. *Active Learning: 101 Strategies To Teach Any Subject.* ERIC, 1996.

[50] Steffen Staab and Rudi Studer. *Handbook on ontologies*. Springer, 2009.

[51] Boontawee Suntisrivaraporn. The CEL manual - version 1.0.

[52] Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, and Boanerges Aleman-meza. OntoQA: Metric-based ontology quality analysis. In *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005.

[53] Carlos Tejo-Alonso, Diego Berrueta, Luis Polo, and Sergio Fernández. Current practices and perspectives for metadata on web ontologies and rules. *International Journal of Metadata, Semantics and Ontologies*, 7(2):93–100, 2012.

[54] Keith J Whittington. Infusing active learning into introductory programming courses. *Journal of Computing Sciences in Colleges*, 19(5):249–259, 2004.

[55] Wei Yu, Qing Li, Junpeng Chen, and Jiaheng Cao. OS_RANK: Structure analysis for ontology ranking. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 339–346. IEEE, 2007.

Glossary

ABox Assertional Box. 9

ALC Attributive Language with Complements. 10

CQ Competency Question. 16

DL Description Logic. 10

GCI General Concept Inclusion. 22

KB Knowledge Base. 9

KRSS Knowledge Representation System Specification. 10

nRQL new Racer Query Language. 16

ODP Ontology Design Pattern. 54

OWA Open World Assumption. 43

OWL Web Ontology Language. 9

RacerPro Renamed ABox and Concept Expression Reasoner Professional. 8

RDF Resource Description Framework. 18

SPARQL Simple Protocol and RDF Query Language. 16

TBox Terminological Box. 9

UNA Unique Name Assumption. 44