Common Framework for Representing Ontologies

Deliverable TONES-D08

D. Calvanese[1], B. Cuenca Grau[3], G. De Giacomo[2], E. Franconi[1], I. Horrocks[3], A. Kaplunova[5], D. Lembo[2], M. Lenzerini[2], C. Lutz[4], D. Martinenghi[1], R. Möller[5], R. Rosati[2], S. Tessaris[1], A.-Y. Turhan[4]

[1] Free University of Bozen-Bolzano
[2] Università di Roma "La Sapienza"
[3] The University of Manchester
[4] Technische Universität Dresden
[5] Technische Universität Hamburg-Harburg

Project:	FP6-7603 – Thinking ONtologiES (TONES)
Workpackage:	WP2– Common Logical Framework for Representing Ontologies
Lead Participant:	Free University of Bozen-Bolzano
Reviewer:	Franz Baader
Document Type:	Deliverable
Classification:	Public
Distribution:	TONES Consortium
Status:	Draft
Document file:	D08_CommonFramework.pdf
Version:	2.1
Date:	August. 25, 2006
Number of pages:	68

Abstract

In this document, we present the general framework for the representation of ontologies that has been designed within TONES as a semantic infrastructure capturing the different formalizations of ontologies as well as their services and the different contexts in which ontologies are used. Then, we illustrate several meaningful instantiations of the framework through ontology based formalisms that have been proposed recently in the literature. Some of these instantiations constitute themselves significant contributions in terms of logic-based ontology formalisms that have been developed within the TONES consortium, and that have been presented recently at high quality scientific venues.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	July 5, 2006	First draft for teleconference discussion
v.1.1	July 11, 2006	Second draft for teleconference
v.1.2	July 12, 2006	Results of teleconference discussion considered
v.2.0	August 15, 2006	First internal release
v.2.1	August 25, 2006	Final version

Contents

1 Introduction

In this document, we present the general framework for the representation of ontologies that has been designed within TONES as a semantic infrastructure capturing the different formalizations of ontologies as well as their services and the different contexts in which ontologies are used.

In the TONES framework, there is no commitment towards a specific representation language or family of languages, neither at the syntactic nor at the semantic level. Rather, the framework provides the means for defining *ontology formalism* according to different needs and with specific properties with respect to both syntax and semantics. This allows one to be sufficiently general to capture virtually all ontology languages, provided that they are equipped with a formal semantics. Indeed, a committed choice of the TONES framework is that it is based on logic, so as to capture the proposals in the literature for formally representing ontologies that are equipped with a formal, logic-based semantics. This choice is motivated by the commonly accepted fact that a logical underpinning of ontologies on the one hand provides a well-defined semantics and hence a common understanding of the meaning of an ontology, and, on the other hand, allows to rephrase certain useful system services of ontology editors and browsers in terms of logical reasoning problems. This, in turn, enables one to base the above mentioned system services on automated reasoning procedures, which is precisely the declared objective, providing added value, of the TONES project.

More specifically, the framework essentially relies on first-order logic and, consequently, the semantics of ontologies is defined in terms of first-order models. However, the framework is open towards different mechanism for selecting models.

It is worth pointing out that the TONES framework has been designed for ontologies, not just generic logic-based formalisms. As such, the ontology formalisms that can be defined within the framework are characterized by a number of distinctive features that we now briefly summarize:

Intensional vs. extensional level. Relying on the experience gained with the study of logic-based formalisms for the representation of structured knowledge, the framework makes an explicit distinction between an *intensional* and an *extensional* level. At the intensional level, general, typically universally quantified knowledge is represented. At the extensional level, specific facts about the modeled domain are captured. Following the tradition of Description Logics [BCM+03], the two levels are called TBox and ABox, respectively.

Unary vs. n-ary predicates. Practically all conceptual modeling and ontology representation formalisms, rely on structuring the domain of interest into sets of elements (either abstract objects or concrete values – see later) with common properties. Unary predicates, which denote such sets of elements, play a prominent role in ontology modeling, and thus are represented explicitly in the TONES framework. In our setting, unary predicates are called *concepts* and *domains* – see below for the distinction between the two (in other settings they are also called classes, entity types, or types). Additionally, also predicates of arbitrary arity are considered, denoting relations of different kinds that

are used to represent the properties of objects belonging to concepts. In our setting, such predicates are called *relations*, *value-relations*, or *attributes* (in other settings they are called also as relationships or associations).

Language for representing ontologies. On the other hand, the TONES framework makes no commitment towards a specific language for representing ontologies, neither at the intensional nor at the extensional level. The language for representing an ontology is a specific parameter of an instantiation of the framework. Typically it will comprise at least a TBox-language, for specifying the intensional component of an ontology, and an ABox-language, for specifying the extensional component. Such languages will in general be subdivided in several sub-languages for specifying different components that may appear in an ontology.

Abstract objects vs. values. At the semantic level, there is an explicit separation between *abstract objects* and *values*: the properties of abstract objects are specified using the modeling constructs of the ontology representation formalism itself; values, instead, belong to predetermined domains (such as that of strings, integers, etc.) whose properties are given a priori, and are used for specifying properties of abstract objects.

Such a distinction is also reflected on the ontology formalisms that can be defined in the TONES framework, where we make use of different alphabets and, correspondingly, of different categories of syntactic elements:

- *object-constants* and *value-constants*, for constants of the two forms;
- *concepts* and *value-domains*, denoting sets of elements of the two forms;
- *relations* and *value-relations*, denoting relationships among elements of the two forms;
- *attributes*, denoting relationships between tuples of abstract objects and values.

Note that the distinction between abstract objects and values has often been blurred in abstract knowledge representation formalisms, such as Description Logics, while it is typically present in practice-oriented ontology formalisms, such as OWL [BvHH+04, SWM04], and is implicit in conceptual modeling formalism. For example, in the Entity-Relationship model [Che76, BCN92], the instances of entities are abstract objects, whose properties are represented within an Entity-Relationship diagram both through relationships with other abstract objects and through attributes that have values belonging to concrete domains such as that of strings, integers, etc..

Ontology services. The TONES framework is tailored to easily describe the mechanisms through which an ontology interacts with the external world, by providing an explicit representation of the services that are offered by an ontology based system to everyone who needs to interact with an ontology, including its designers, maintainers, and end users.

In the framework, each *service* applied to ontologies comes with:

- a *signature*, specifying the name of the service and its arguments;
- a set of *preconditions*, specifying the conditions that the arguments have to satisfy in order for the service to be invoked;
- a meaning, i.e., a specification of the result of the service invocation for every possible combination of the arguments satisfying the preconditions.

The domain of the result of a service invocation may be structured into different components, to take into account services that return complex results. Specifically, services that manipulate an ontology, possibly modifying it, are taken into account by including the modified ontology itself as (part of) the result of the service invocation. In other words, we take a purely functional view of ontology services[1].

Among the services, we distinguish *reasoning services*, which are those services for which the result of the invocation is independent of the syntactic representation of the ontology to which the service is applied. In other words, a reasoning service will provide the same results when invoked on two ontologies with different syntactic representations but with the same sets of models. A typical example of a reasoning service is the one that asks "Is concept C consistent in an ontology?", which requires to check whether there is at least one model of the ontology in which the concept C is interpreted as a non-empty set. An example of a service that is not a reasoning service is the one that requires to "Return the set of all concept names appearing in the ontology". More in general, services that involve forms of meta-querying, i.e., querying about the syntactic structure of an ontology, are not reasoning services.

At the level of the framework, we do not distinguish between *basic services*, i.e., services that do not rely on other services, and *composite services*, that instead rely on other services for their realization. However, in a specific instantiation of the TONES framework such a distinction may be made. In such a case, the choice of whether a service is considered as basic or composite may be based purely on the specification of the service, or may also involve considerations about efficiency in the realization of the service.

Stand-alone ontologies vs. ontologies in a wider context. In the TONES framework, besides stand-alone ontologies, we take into account that ontologies may be placed in a wider context and interact with other systems. Such systems may be either external systems, of which we do not have or want to provide a complete specification of syntax and semantics, or other ontologies, itself modeled in the framework. Hence, the TONES framework explicitly considers three contexts of application of ontologies:

- A *stand-alone ontology*, which is essentially defined as a pair $\langle \text{TBox}, \text{ABox} \rangle$, whose semantics is specified in terms of *FOL interpretations* over its components, and which offers a set of services.
- A *situated ontology*, i.e., an ontology interacting with an external system that is not fully modeled within the framework. Rather, the interaction between the ontology

[1] Note that this concerns only the abstract representation of services within the TONES framework. Obviously, services will in general *not* be realized in a purely functional way.

and the external system is represented by a mapping, and the state of the external system is represented only as far as it affects the mapping, and through the mapping constrains the possible models of the ontology. Services for situated ontologies may include, besides services that deal with the ontology itself, also services that, through the ontology, access and possibly modify the external system.

A typical example of an external system interacting with an ontology is a database, storing and providing (part of) the extensional information for an ontology.

- A system of *peer ontologies*, i.e., a set of ontologies with a set of pairwise mappings between them. The entire network of peer ontologies and mappings is seen as a whole system constraining the models of the ontologies. Note that, from the point of view of a specific ontology $\mathcal{O}$ in the system of peer ontologies, the other ontologies may be viewed as external systems with which $\mathcal{O}$ interacts and that constrain the possible models of $\mathcal{O}$. However, in the TONES framework, we have chosen to take into account the case of peer ontologies separately from the case of situated ontologies for various reasons, which we briefly illustrate:
  - In the case of a situated ontology, the external system is modeled only as far as it affects the mappings to the ontology, whereas in a system of peer ontologies all ontologies are fully modeled through an ontology formalism that is compatible with the TONES framework.
  - The semantics of a system of peer ontologies may be specified in a more complex way than by a simple composition of the semantics of a single ontology and the direct mappings such an ontology has to other ontologies. Indeed, the semantics of an ontology $\mathcal{O}$ in a system of peer ontologies, and hence of the whole system, may depend on all ontologies involved in the system rather than only on the ones to which $\mathcal{O}$ is directly connected through mappings, and such a situation could not properly be taken into account by the notion of situated ontology.
  - Each single ontology in a system of peer ontology may in fact be connected also to another system, and hence represented as a situated ontology rather than a stand-alone ontology.

Structure of the document

We present now, in Section 2, the TONES ontology framework. Then, in the rest of the document, we illustrate several meaningful instantiations of the framework through ontology based formalisms that have been proposed recently in the literature. Specifically, we provide the following instantiations of the various components of the framework:

- We provide several instantiations of the framework for stand-alone ontologies: an instantiation through standard description logics (Section 3); an instantiation through OWL 1.1, the latest version of the W3C standard Web Ontology Language (OWL), which is currently under consideration as the next standard ontology language for

the W3C (Section 4; an instantiation through the rule-based language SWRL (Section 5).

- We provide an instantiation of the framework for situated ontologies through *DL-Lite*, a recent proposal for a description logic specifically tailored to provide ontology-bases access to data stored in relational databases (Section 6).
- We provide two instantiation of the framework for peer ontologies based on different semantic specifications for the mappings between ontologies, and hence for the whole system.

We point out that the instantiations we provide, on the one hand, serve the purpose to illustrate through examples how the TONES ontology framework can be instantiated to capture meaningful ontology-based formalisms. On the other hand, some of these instantiations constitute themselves significant contributions in terms of logic-based ontology formalisms that have been developed within the TONES consortium, and that have been presented recently at high quality scientific venues [HKS06, HPSBT05, CDGL$^+$06a, CDGL$^+$06b].

2 Ontology framework

In this section, we present the TONES framework for representing ontologies. The framework makes an explicit distinction between stand-alone ontologies, situated ontologies, and peer ontologies. In each of the three cases, the framework determines the form of an *ontology formalism* (respectively for stand-alone, situated, and peer ontologies), which is itself a mechanisms for defining ontologies. Moreover, the framework determines how *services* offered by (stand-alone, situated, and peer) ontologies should be specified.

We now describe separately the three cases of stand-alone ontologies (Subsection 2.1), situated ontologies (Subsection 2.2), and peer ontologies (Subsection 2.3), and in each of the three cases we define first the form of an ontology formalism, then we describe how ontologies in an formalism are specified (w.r.t. both syntax and semantics), and finally we describe ontology services.

2.1 Framework for stand-alone ontologies

We describe now the structure of a *stand-alone ontology*, i.e., an ontology considered as an independent system that maintains intensional and extensional information, and provides services to access and manipulate such information.

2.1.1 Ontology formalism

An ontology formalism provides the specification of both the syntax and the semantics of ontologies for that formalism. Formally, an *ontology formalism* is a five-tuple $\mathcal{F} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, Sem \rangle$, where:

- Σ, $\mathcal{L}_C$, $\mathcal{L}_T$, $\mathcal{L}_A$, are respectively an alphabet, a concept language, a TBox language and an ABox language, specifying the syntax of ontologies defined in $\mathcal{F}$, and

- *Sem* is a *semantic specification*, specifying the semantics of ontologies defined in $\mathcal{F}$.

We describe the various elements of $\mathcal{F}$, treating first the syntactic and then the semantic aspects.

Specification of the syntax of ontologies. The first four elements Σ, $\mathcal{L}_C$, $\mathcal{L}_T$, and $\mathcal{L}_A$ of $\mathcal{F}$, described below, provide the mechanisms for defining the syntax of ontologies defined in $\mathcal{F}$.

- Σ is a finite or countably infinite alphabet, partitioned into the following sub-alphabets:
  - Σ_c of *concept names*,
  - Σ_r of *relation names*,
  - Σ_d of *value-domain names*,
  - Σ_u of *value-relation names*,
  - Σ_a of *attribute names*,
  - Σ_o of *object-constants*,
  - Σ_v of *value-constants*.

  Each symbol in Σ_r or Σ_u has an associated *arity* $n \geq 2$. Each symbol in Σ_a has an associated *arity* $n \geq 1$.

- $\mathcal{L}_C$ is a *concept language*, i.e., a language for expressions of concepts, relations, domains, and attributes. We assume that expressions of such language are built over the alphabets Σ_c, Σ_r, Σ_d, Σ_u, and Σ_a only, and that they do not make use of the alphabets Σ_o and Σ_v of constants.

- $\mathcal{L}_T$ is a *TBox language*, i.e., a language for expressing intensional knowledge, making use of expressions in the language $\mathcal{L}_C$. An element of $\mathcal{L}_T$ is called a *TBox*.

- $\mathcal{L}_A$ is an *ABox language*, i.e., a language for expressing extensional knowledge, making use of expressions in the language $\mathcal{L}_C$ and of constants. An element of $\mathcal{L}_A$ is called an *ABox*.

Several remarks on the syntactic elements of $\mathcal{F}$ are in order.

- The assumption of the various sub-alphabets of Σ to be disjoint is made to simplify the formal treatment. In concrete instantiations of the framework, from the point of view of users, it may be possible to have non-disjoint alphabets. In this case we would assume that, from the point of view of the ontology-based system, symbols are prefixed with some distinguishing prefix, e.g., "concept:" for concepts, etc.

- Expressions of the concept language $\mathcal{L}_C$ will in general be constructed by making use of *constructors* that are specific of the concept language, and that allow for a compositional syntax for the various kinds of expressions.

- In specific cases, $\mathcal{L}_C$ may foresee expressions that contain variables. Note, however, that we do not include explicitly in Σ an alphabet of variables, since Σ contains only those symbols that appear in interpretative structures (i.e., FOL structures – see below), and in these structures we do not need to mention variables. Indeed, variables are not interpreted per see but are assigned a semantics by means of variable assignments. Of course, for convenience, specific instantiations of the framework may add variables to the syntax, as a new syntactic category, with a corresponding alphabet of variables.

- In general, a TBox of $\mathcal{L}_T$ will be constituted by a set of universally quantified assertions, e.g., inclusion assertions between concepts or relations, but the framework does not rule out other forms of intensional knowledge.

- In general, an ABox of $\mathcal{L}_A$ will be constituted by a set of facts, expressing knowledge about instances of concepts, relations, etc., but again the framework does not rule out other forms of extensional knowledge, e.g., existentially quantified assertions.

Specification of the semantics of ontologies. In the TONES framework, the semantics of ontologies is specified in terms of FOL interpretations for the alphabet Σ. In our setting, a *FOL interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where:

- $\Delta^{\mathcal{I}}$ is an *interpretation domain*, partitioned into a set $\Delta_o^{\mathcal{I}}$ of *abstract objects* and a set $\Delta_v^{\mathcal{I}}$ of *values*;

- $\cdot^{\mathcal{I}}$ is an *interpretation function* that assigns
  - to each concept $C \in \Sigma_c$, a set $C^{\mathcal{I}} \subseteq \Delta_o^{\mathcal{I}}$,
  - to each relation $R \in \Sigma_r$ of arity n, a set $R^{\mathcal{I}} \subseteq (\Delta_o^{\mathcal{I}})^n$,
  - to each value-domain $D \in \Sigma_d$, a set $D^{\mathcal{I}} \subseteq \Delta_v^{\mathcal{I}}$,
  - to each value-relation $U \in \Sigma_u$ of arity n, a set $U^{\mathcal{I}} \subseteq (\Delta_v^{\mathcal{I}})^n$,
  - to each attribute $A \in \Sigma_a$ of arity n a set $A^{\mathcal{I}} \subseteq (\Delta_o^{\mathcal{I}})^n \times \Delta_v^{\mathcal{I}}$,
  - to each object-constant $a \in \Sigma_o$ an element $a^{\mathcal{I}} \in \Delta_o^{\mathcal{I}}$,
  - to each value-constant $v \in \Sigma_v$ an element $v^{\mathcal{I}} \in \Delta_v^{\mathcal{I}}$.

Note that, at the semantic level, there is an explicit separation between *abstract objects* and *values*: the properties of abstract objects are specified using the modeling constructs of the ontology representation formalism itself; values, instead, belong to predetermined domains (such as that of strings, integers, etc.) whose properties are given a priori, and are used for specifying properties of abstract objects.

In an ontology formalism $\mathcal{F} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, Sem \rangle$, the last component *Sem* is a *semantic specification*, which determines how to assign semantics to ontologies defined in $\mathcal{F}$. Formally, *Sem* is a triple $(\mathcal{S}, \delta, \odot)$, where:

- $\mathcal{S}$ is a set of FOL interpretations for Σ, called *admissible* interpretations. Such a set is intended to take into account semantic assumptions that may be present and that constrain FOL interpretations (e.g., the unique-name assumption).

- δ provides a mechanism for assigning, given an admissible FOL interpretation, a truth value to each TBox and ABox. Formally, δ is a function, that, given a FOL interpretation $\mathcal{I} \in \mathcal{S}$ and a TBox or an ABox $\mathcal{B} \in \mathcal{L}_T \cup \mathcal{L}_A$, returns a truth value $\delta(\mathcal{I}, \mathcal{B}) \in \{\mathsf{true}, \mathsf{false}\}$. We also use $\mathcal{B}^{\mathcal{I}}$ to denote $\delta(\mathcal{I}, \mathcal{B})$, and when $\delta(\mathcal{I}, \mathcal{B}) = \mathsf{true}$ we say that $\mathcal{I}$ *satisfies* $\mathcal{B}$.

- $\odot$ is a binary operation on sets of FOL interpretations, typically set intersection, that specifies how to compose the interpretations that satisfy the TBox with those that satisfy the ABox.

2.1.2 Ontology

Given an ontology formalism $\mathcal{F} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, Sem \rangle$, an *$\mathcal{F}$-ontology* (or simply *ontology*, when $\mathcal{F}$ is clear from the context) is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where:

- $\mathcal{T}$ is a *TBox*, which is a sentence in the language $\mathcal{L}_T$;

- $\mathcal{A}$ is an *ABox*, which is a sentence in the language $\mathcal{L}_A$;

Given an $\mathcal{F}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, the semantic specification $Sem = (\mathcal{S}, \delta, \odot)$ of $\mathcal{F}$ determines the semantics of $\mathcal{O}$ by specifying the models of $\mathcal{O}$. Intuitively, a model of $\mathcal{O}$ is a FOL interpretation in $\mathcal{S}$ that satisfies both the TBox and the ABox of $\mathcal{O}$. Formally, the set of *$\mathcal{F}$-models* (or simply *models*, when $\mathcal{F}$ is clear from the context) of $\mathcal{O}$ is $Mod_{\mathcal{F}}(\mathcal{O}) = Mod_{\mathcal{F}}(\mathcal{T}) \odot Mod_{\mathcal{F}}(\mathcal{A})$, with

$$\begin{array}{rcl} Mod_{\mathcal{F}}(\mathcal{T}) & = & \{\ \mathcal{I} \in \mathcal{S} \ \mid\ \delta(\mathcal{I}, \mathcal{T}) = \mathsf{true}\ \} \\ Mod_{\mathcal{F}}(\mathcal{A}) & = & \{\ \mathcal{I} \in \mathcal{S} \ \mid\ \delta(\mathcal{I}, \mathcal{A}) = \mathsf{true}\ \} \end{array}$$

As mentioned, typically, $\odot$ will compute just the intersection of the set of interpretations that satisfy $\mathcal{T}$ with the set of interpretations that satisfy $\mathcal{A}$. However, the framework allows also for the case where the two sets of interpretations need to be composed in a more complex way. For example, in specific cases, even when the intersection of $Mod_{\mathcal{F}}(\mathcal{T})$ and $Mod_{\mathcal{F}}(\mathcal{A})$ is empty (i.e., the TBox and the ABox are mutually inconsistent), one may nevertheless want to consider a non-empty set of models for the ontology.

2.1.3 Ontology services

An ontology formalism $\mathcal{F}$ is equipped with a set of services, providing the functionalities that $\mathcal{F}$-ontologies offer to users. More precisely, given an ontology formalism $\mathcal{F}$, an *$\mathcal{F}$-ontology service* (or simply, $\mathcal{F}$-service, or service, when $\mathcal{F}$ is clear from the context) S is specified as follows.

- At the syntactic level, the service has

  - a *signature*, i.e., a name, an arity $n \geq 0$ specifying the number of arguments of S, and n domains specifying the types of the n arguments; in addition to the n arguments of the service, there is always a distinguished argument which is the $\mathcal{F}$-ontology to which the service is applied;

  - a set of *preconditions*, each of which is a subset of the cross-product of the domains of all arguments (including the distinguished ontology argument);
  - a domain for the result.

- The service is given a *meaning*, which is a specification of what is the result of the service invocation for every given ontology and every possible combination of the arguments that satisfy the preconditions.

For an $\mathcal{F}$-service S, an $\mathcal{F}$-ontology $\mathcal{O}$, and a tuple $\vec{p}$ of additional arguments to S such that $\mathcal{O}$ and $\vec{p}$ satisfy the preconditions of S, we denote with $S(\mathcal{O}, \vec{p})$ the result of the application of S to $\mathcal{O}$ and $\vec{p}$.

Among the services, we distinguish a particular class, called reasoning services, which have the property of being independent of the syntactic representation of the ontology to which they are applied. Formally, an $\mathcal{F}$-service S is called a *reasoning service* if, for every pair $\mathcal{O}_1$, $\mathcal{O}_2$ of $\mathcal{F}$-ontologies such that $Mod_{\mathcal{F}}(\mathcal{O}_1) = Mod_{\mathcal{F}}(\mathcal{O}_2)$, and for every tuple $\vec{p}$ of additional arguments for S such that $\mathcal{O}_1$ and O_2 together with $\vec{p}$ satisfy the preconditions of S, we have that $S(\mathcal{O}_1, \vec{p}) = S(\mathcal{O}_2, \vec{p})$.

Notable examples of reasoning services are:

- ontology satisfiability;
- logical implication of an assertion, given an ontology;
- query answering;
- update of an ontology;
- computing the least-common subsumer;
- classification.

Examples of services that are not reasoning services are those that involve forms of meta-querying, i.e., querying about the syntactic structure of an ontology. For example, providing the list of concept names mentioned in the TBox, or the list of concepts of which a given individual is an instance.

2.2 Situated ontologies

A *situated ontology* represents a (stand-alone) ontology situated in a more general system, here called *environment*. In the TONES framework, the environment itself is not modeled; rather, the interaction between the ontology and the environment is represented by a mapping. Hence, in defining situated ontologies for an environment $\mathcal{E}$, we refer to an alphabet $\Sigma_{\mathcal{E}}$ of symbols, disjoint from Σ, to be used together with Σ in a language $\mathcal{L}_{\mathcal{E}}$ of mappings between ontologies and the environment. From a semantical point of view, the mapping constrains the possible models of the ontology according to the state of the environment, which acts as a parameter.

2.2.1 Situated ontology formalism

Given an environment $\mathcal{E}$, a *situated ontology formalism (for $\mathcal{E}$)* is a tuple $\mathcal{F}_\mathcal{E} = \langle \mathcal{F}, \Sigma_\mathcal{E}, \mathcal{L}_\mathcal{E}, Sem_\mathcal{E} \rangle$, where:

- $\mathcal{F} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, Sem \rangle$ is an ontology formalism;
- $\Sigma_\mathcal{E}$ is an alphabet that is disjoint from Σ;
- $\mathcal{L}_\mathcal{E}$ is a language over $\Sigma \cup \Sigma_\mathcal{E}$ for the mapping between $\mathcal{F}$-ontologies and the environment in which the $\mathcal{F}$-ontologies are situated; the form of such a language will obviously depend on the environment;
- $Sem_\mathcal{E}$ is a *semantic specification for the mappings*, i.e., a triple $(\mathcal{S}_\mathcal{E}, \delta_\mathcal{E}, \odot_\mathcal{E})$, where:
  - $\mathcal{S}_\mathcal{E}$ is a set of *states* for the environment $\mathcal{E}$,
  - $\delta_\mathcal{E}$ is a function that, given an $\mathcal{F}$-ontology $\mathcal{O}$, a mapping $\mathcal{M}$ in $\mathcal{L}_\mathcal{E}$, a FOL interpretation $\mathcal{I}$ for $\mathcal{O}$, and a *state* $E \in \mathcal{S}_\mathcal{E}$, returns a truth value $\delta_\mathcal{E}(\mathcal{O}, \mathcal{M}, \mathcal{I}, E) \in \{\mathsf{true}, \mathsf{false}\}$. Intuitively, a FOL interpretation $\mathcal{I}$ such that $\delta_\mathcal{E}(\mathcal{O}, \mathcal{M}, \mathcal{I}, E) = \mathsf{true}$ is considered to be *coherent* (according to the mappings) with $\mathcal{O}$ and with the state E of the environment;
  - $\odot_\mathcal{E}$ is a binary operation on sets of FOL interpretations, typically set intersection, that specifies how to compose the interpretations that are models of an $\mathcal{F}$-ontology (considered as a stand-alone ontology) with those that are coherent with a mapping.

2.2.2 Situated ontology

Given a situated ontology formalism $\mathcal{F}_\mathcal{E} = \langle \mathcal{F}, \Sigma_\mathcal{E}, \mathcal{L}_\mathcal{E}, Sem_\mathcal{E} \rangle$, an *$\mathcal{F}_\mathcal{E}$-situated ontology* (or simply *situated ontology*, when $\mathcal{F}_\mathcal{E}$ is clear from the context) is a pair $\mathcal{O}_\mathcal{E} = \langle \mathcal{O}, \mathcal{M} \rangle$ where:

- $\mathcal{O}$ is an $\mathcal{F}$-ontology;
- $\mathcal{M}$ is a *mapping* to the environment, specified in the language $\mathcal{L}_\mathcal{E}$.

Given an $\mathcal{F}_\mathcal{E}$-situated ontology $\mathcal{O}_\mathcal{E} = \langle \mathcal{O}, \mathcal{M} \rangle$ and a state E of the environment, the semantic specifications Sem for $\mathcal{F}$-ontologies and $Sem_\mathcal{E}$ for the mappings determine the semantics of $\mathcal{O}_\mathcal{E}$ by specifying the models of $\mathcal{O}_\mathcal{E}$, i.e., the models of $\mathcal{O}$ that are also coherent according to $\mathcal{M}$ with $\mathcal{O}$ and E.

Formally, let the semantic specification of $\mathcal{F}$ be $Sem = (\mathcal{S}, \delta, \odot)$, with $\mathcal{S}$ the set of admissible interpretations for $\mathcal{F}$. The set of models that are *coherent according to $\mathcal{M}$ with $\mathcal{O}$ and E* are

$$Mod_\mathcal{E}(\mathcal{O}, \mathcal{M}, E) \;=\; \{\; \mathcal{I} \in \mathcal{S} \;\mid\; \delta_\mathcal{E}(\mathcal{O}, \mathcal{M}, \mathcal{I}, E) = \mathsf{true} \;\}$$

Then, the set of *$\mathcal{F}_\mathcal{E}$-models* (or simply *models*, when $\mathcal{F}_\mathcal{E}$ is clear from the context) *of $\mathcal{O}_\mathcal{E}$ with respect to E* is

$$Mod_{\mathcal{F}_\mathcal{E}}(E, \mathcal{O}_\mathcal{E}) \;=\; Mod_\mathcal{F}(\mathcal{O}) \odot_\mathcal{E} Mod_\mathcal{E}(\mathcal{O}, \mathcal{M}, E)$$

As mentioned, typically $\odot_{\mathcal{E}}$ will compute just the intersection of the models of $\mathcal{O}$ with the set of interpretations that are coherent according to $\mathcal{M}$ with $\mathcal{O}$ and E. However, the framework allows for taking into account also more complex situations, e.g., the one that would result from cleaning operations on the information retrieved through the mappings.

A prominent example for a situated ontology is an ontology that accesses an external database to extract values for its attributes and, indirectly, also objects for its concepts and relations.

2.2.3 Services for situated ontologies

A situated ontology exhibits the same kind of services as a stand-alone ontology. However, the services may additionally refer with suitable parameters to the environment.

Again, we consider among the services for situated ontologies a distinguished class of services called *reasoning services*, which are independent of the syntactic specification of the ontology.

An example of a services for a situated ontology is the one where queries over an ontology are answered by accessing an underlying database.

2.3 Peer ontologies

Peer ontologies are a set of (possibly situated) ontologies and a set of pairwise mappings between them. We define the semantics in such a way that the entire network of peer ontologies and mappings specified between them is seen as a whole system that constrains again the models of the various ontologies. This choice allows our framework to be as general as possible, and thus to capture different possible semantic interpretations of peer ontologies.

2.3.1 Peer ontologies formalism

Formally, a *peer ontologies formalism* is a triple $\mathcal{F}_P = \langle \{\mathcal{F}^i\}_{1 \leq i \leq k}, \{\mathcal{L}_M^{ij}\}_{1 \leq i,j \leq k, i \neq j}, Sem_P \rangle$, where:

- $\{\mathcal{F}^i\}_{1 \leq i \leq k}$ is a sequence of (possibly situated) ontology formalisms with pairwise disjoint alphabets;
- $\{\mathcal{L}_M^{ij}\}_{1 \leq i,j \leq k, i \neq j}$ is a sequence of *mapping languages*, where each $\mathcal{L}_M^{ij}$ is a language for the peer mappings from the $\mathcal{F}^i$-ontologies to $\mathcal{F}^j$-ontologies, expressed over the alphabet $\Sigma^i \cup \Sigma^j$, where Σ^i and Σ^j are respectively the alphabets of $\mathcal{F}^i$ and $\mathcal{F}^j$. Notice that we require $i \neq j$, since mappings have to be established only between different ontologies. In the following, we will always assume $i \neq j$, even if not specified ;
- Sem_P is a *semantic specification for peer ontologies*, which is a pair $(\mathcal{S}_P, \delta_P)$, where:
  - $\mathcal{S}_P$ is a set of interpretation structures for the language $\mathcal{L}_P$ for the specification of systems of $\mathcal{F}_P$-peer ontologies (formally defined below). Hence, each element

of $\mathcal{L}_P$ is a specification of a system of $\mathcal{F}_P$-peer ontologies, which informally consists of one ontology $\mathcal{O}^i$ for each $i \in \{1, \ldots, k\}$ and one peer mapping specification for each $\mathcal{L}_M^{ij}$;

– δ_P provides a mechanism for assigning, given an interpretation structure in $\mathcal{S}_P$, a truth value to each system of peer ontologies. Formally, δ_P is a function that, given an interpretation structure $S_P \in \mathcal{S}_P$ and a system $\mathcal{P}$ of peer ontologies, i.e., an element of $\mathcal{L}_P$, returns a truth value $\delta_P(S_P, \mathcal{P}) \in \{\mathsf{true}, \mathsf{false}\}$.

Note that, in specifying the semantics of a system of peer ontologies, the TONES framework foresees arbitrary interpretation structures, rather than restricting the possibility to FOL structures. This provides more flexibility in the combination of the semantics for the single ontologies that participate to a system of peer ontologies, and allows one to take into account also situations where the overall semantics cannot be properly represented as a flat FOL theory.

2.3.2 Peer ontologies

Consider a peer ontologies formalism $\mathcal{F}_P = \langle \{\mathcal{F}^i\}_{1 \leq i \leq k}, \{\mathcal{L}_M^{ij}\}_{1 \leq i,j \leq k, i \neq j}, Sem_P \rangle$. A *system of $\mathcal{F}_P$-peer ontologies* (or simply *peer ontologies*, when $\mathcal{F}_P$ is clear from the context) is a pair $\mathcal{P} = \langle \{\mathcal{O}^i\}_{1 \leq i \leq k}, \{\mathcal{M}^{ij}\}_{1 \leq i,j \leq k, i \neq j} \rangle$, where:

- each $\mathcal{O}^i$ is an $\mathcal{F}^i$-ontology;
- each $\mathcal{M}^{ij}$ is a peer mapping from $\mathcal{O}^i$ to $\mathcal{O}^j$ expressed in the language $\mathcal{L}_M^{ij}$ (obviously, $i \neq j$ for each $\mathcal{M}^{ij}$).

Given a system $\mathcal{P} = \langle \{\mathcal{O}^i\}_{1 \leq i \leq k}, \{\mathcal{M}^{ij}\}_{1 \leq i,j \leq k, i \neq j} \rangle$ of $\mathcal{F}_P$-peer ontologies, the semantic specification for peer ontologies Sem_P determines the semantics of the system $\mathcal{P}$, by specifying the models of $\mathcal{P}$: An *$\mathcal{F}_P$-model* of $\mathcal{P}$ is an interpretation $\mathcal{M} \in \mathcal{S}_P$ such that $\delta_P(\mathcal{M}, \mathcal{P}) = \mathsf{true}$.

Prominent examples for peer ontologies are:

- an ontology importing knowledge from another one through a set of mappings;
- a peer-to-peer systems of ontologies, in which each ontology imports and exports knowledge from the other ones through possibly cyclic mappings.

2.3.3 Services for peer ontologies

An ontology that is part of a set of peer ontologies exhibits the same kind of services as a stand-alone (resp., situated) ontology. Additionally, a set of peer ontologies may exhibit additional services that refer to more than one ontology.

An example of service for peer ontologies is the integration of two ontologies to produce a new one.

3 Standard description logics

Description logics (DLs) provide a logic-based formalism to represent a conceptual knowledge about domain of interest and to infer implicit information from explicitly made assertions in the knowledge base. In the context of Semantic Web, which is a research topic of current importance, DLs have a considerable impact on building ontologies equipped with model-theoretic semantics and a range of tractable reasoning services. Standard ontology modeling languages such as OWL Lite, OWL DL [HPSvH03, BvHH+04, PSHH04, SWM04], and OWL 1.1 (see Section 4) are based on expressive DLs (respectively $\mathcal{SHIF}$, $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$). Therefore, in the literature, the notion of "ontology" is often used as a synonym for a description logic knowledge base. However, in this section, we talk about the description logics formalism as a (quite natural) instantiation of the stand-alone ontology framework which was introduced in Section 2. We describe now in detail how syntax and semantics of standard DLs as well as inference services correspond to the components of this framework.

3.1 The description logic $\mathcal{ALC}$

We start with a short overview of main features of description logics on the example of $\mathcal{ALC}$, the core language of the expressive DL family (for details see [BCM+03]). Then, we dwell on some additional language constructs, available in many specific description logics. In parallel, we present the syntax and semantics of selected DLs in terms of a stand-alone ontology formalism.

Definition 3.1 ($\mathcal{ALC}$ syntax) Given finite and mutually disjoint sets $\mathcal{A}$ of concept names, $\mathcal{R}$ of role names , and $\mathcal{O}$ of (abstract) individual names, the set of $\mathcal{ALC}$ concepts is inductively defined as follows:

1. every concept name $A \in \mathcal{A}$ is a concept;
2. if C and D are concepts and $R \in \mathcal{R}$ is a role name, then the following expressions are also concepts:

   $\neg C$ (negation)
   $C \sqcap D$ (conjunction)
   $C \sqcup D$ (disjunction)
   $\exists R.C$ (existential restriction)
   $\forall R.C$ (value restriction)

The expressions $\top$ (universal concept) and $\bot$ (unsatisfiable concept) are used as abbreviations for respectively $A \sqcap \neg A$ and $A \sqcup \neg A$, where A is a concept name. ∎

Definition 3.2 (GCI, TBox) If C and D are concepts, then $C \sqsubseteq D$ is called a *terminological axiom* (or generalized concept inclusion, or GCI). A finite set of terminological axioms is called a terminology, or *TBox* for short. ∎

Definition 3.3 (ABox assertion, ABox) If $a, b \in \mathcal{O}$ are individual names, C is a concept and $R \in \mathcal{R}$ is a role name, then the following are *assertional axioms*:

$C(a)$ (concept membership assertion)

$R(a, b)$ (role membership assertions)

A finite set of assertional axioms is called an *ABox*. ∎

Definition 3.4 (Knowledge Base) A knowledge base is a tuple $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox (as defined in Definitions 3.2 and 3.3 respectively).

In the ontology formalism, a DL knowledge base is seen as an instantiation of an *$\mathcal{F}$-ontology*, that is the pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. ∎

Definition 3.5 ($\mathcal{ALC}$ syntax in terms of a stand-alone ontology formalism) An *ontology formalism* is a five-tuple $\mathcal{F} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, Sem \rangle$, where Σ, $\mathcal{L}_C$, $\mathcal{L}_T$ and $\mathcal{L}_A$ respectively are an alphabet, a concept language, a TBox language, an ABox language and a *semantic specification*, as specified in Section 2.

Given prerequisites of Definition 3.1, we assume that

- every concept name $A \in \mathcal{A}$ is an element of the alphabet Σ_c (*concept names*)
- every role name $R \in \mathcal{R}$ is an element of the alphabet Σ_r (*relation names*)
- every (abstract) individual name $i \in \mathcal{O}$ is an element of the alphabet Σ_o (*object constants*)

Summing up, an alphabet Σ of an $\mathcal{ALC}$-based ontology consists of tree disjoint sets: $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_o$. $\mathcal{ALC}$ concepts (see Definition 3.1) constitute the concept language $\mathcal{L}_C$.

Sets of $\mathcal{ALC}$ general inclusion axioms, which express an intensional knowledge, are treated as elements of the TBox language $\mathcal{L}_T$. In other words, all possible $\mathcal{ALC}$-TBoxes (as defined in Definition 3.2) are "words" of the language $\mathcal{L}_T$.

The ABox language $\mathcal{L}_A$ contains as its "words" all possible sets of $\mathcal{ALC}$ assertions, or $\mathcal{ALC}$-ABoxes (see Definition 3.3). ∎

Definition 3.6 ($\mathcal{ALC}$ semantics) The model-theoretic semantics of $\mathcal{ALC}$ is given in the standard form using a Tarskian interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a nonempty set, called the *interpretation domain*, and $\cdot^{\mathcal{I}}$ is the *interpretation function*. The interpretation function assigns

- to each concept name $A \in \mathcal{A}$, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$;
- to each (binary) role name $R \in \mathcal{R}$, a set $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;
- to each individual name $a \in \mathcal{O}$, an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ such that unique-name assumption holds (i.e., different names are mapped to different elements of the domain).

For complex concepts, the semantics is defined as follows:

$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ (complement)

$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ (conjunction)

$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ (disjunction)

$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ (existential restriction)

$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ (value restriction)

A concept C is satisfiable if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. A GCI $C \sqsubseteq D$ is satisfied by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation which satisfies all axioms of a TBox is called a *model* of the TBox. A concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. An ABox $\mathcal{A}$ is satisfied by an interpretation $\mathcal{I}$ if for all concept assertions $C(a) \in \mathcal{A}$ it holds that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and for all role membership assertions $R(a,b) \in \mathcal{A}$ it holds that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. A model of a knowledge base is an interpretation that is a model of $\mathcal{T}$ and $\mathcal{A}$. ∎

Definition 3.7 ($\mathcal{ALC}$ semantics in terms of a stand-alone ontology formalism) An ontology formalism introduces the notion of *semantic specification Sem*, that is a pair $(\mathcal{S}, \delta)$, where $\mathcal{S}$ is a set of *admissible* interpretations and δ is a function. Given an FOL interpretation $\mathcal{I} \in \mathcal{S}$ and a TBox or an ABox $\alpha \in \mathcal{L}_T \cup \mathcal{L}_A$, δ returns a boolean value $\delta(\mathcal{I}, \alpha) \in \{\mathsf{true}, \mathsf{false}\}$.

Each $\mathcal{ALC}$ interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ can be regarded as an FOL interpretation of an $\mathcal{F}$-ontology. If the unique name assumption is imposed, then the semantic specification *Sem* must take into account this requirement. Further, we note, that

- the interpretation domain $\Delta^{\mathcal{I}}$ is equivalent to a set $\Delta^{\mathcal{I}}_o$ of abstract objects,
- the set of values $\Delta^{\mathcal{I}}_v$ is empty, and
- the interpretation function $\cdot^{\mathcal{I}}$ maps
  - each concept name $C \in \Sigma_c$ to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_o$
  - each (binary) relation name $R \in \Sigma_r$ into a set $R^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}}_o)^2$
  - each abstract individual name $a \in \Sigma_o$ into an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_o$

Each $\mathcal{ALC}$ model corresponds to an *$\mathcal{F}$-model*, and the problem of satisfiability of an $\mathcal{ALC}$-knowledge base corresponds to the problem of satisfiability of an $\mathcal{F}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.
∎

3.2 Description logics with additional constructs

3.2.1 Concrete domains

Support for datatypes and values is important for ontology modeling languages. Therefore, a proposed ontology formalism foresees the mapping for concrete values and relations on values. In this subsection, we consider DLs which integrate the ability to operate with so-called concrete domains. Discussing the logic $\mathcal{ALC}(\mathcal{D})$ ([BH91, BH92]), we present the next example how the instantiation of an $\mathcal{F}$-ontology can be defined.

Definition 3.8 (Concrete domain) A *concrete domain* $\mathcal{D}$ is a tuple $(\Delta^{\mathcal{D}}, \Phi)$ of a non-empty set $\Delta^{\mathcal{D}}$ and a set of predicates names Φ. Predicates are defined in a certain language (e.g., linear inequations over polynomials or equations over strings). Each predicate name $P \in \Phi$ is associated with an arity n and an n-ary predicate $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$. Examples of concrete domains are integers, reals, strings or a spatial domain, whereas values such as a number "18" are interpreted as elements of $\Delta^{\mathcal{D}}$.

A concrete domain $\mathcal{D}$ is called *admissible* iff the set of its predicate names is closed under negation and contains a name for $\Delta^{\mathcal{D}}$, and the satisfiability problem for $\mathcal{D}$ is decidable (for details see [BCM$^+$03]). ∎

While $\mathcal{ALC}(\mathcal{D})$ is an extension of the basic logic $\mathcal{ALC}$, we will mention further only its additional properties. In $\mathcal{ALC}(\mathcal{D})$, individuals are separated into two disjoint sets of abstract and concrete objects. With concrete domains, functional roles (so-called *features*) are introduced, which are partial functions that map individuals of the abstract domain to elements of the concrete domain. A composition of features is called a *feature chain*. Further, sets of role and feature names must be disjoint. Relations between concrete objects are described with help of concrete predicates. Moreover, the new concept-forming predicate constructor is added to the set of $\mathcal{ALC}$ concepts.

Definition 3.9 ($\mathcal{ALC}(\mathcal{D})$ syntax) We assume disjoint sets $\mathcal{C}$, $\mathcal{R}$, $\mathcal{F}$, Φ, $\mathcal{O}_A$ and $\mathcal{O}_D$ of concept, role, feature, predicates, abstract and concrete individual names (or individuals for short) respectively. Let $P \in \Phi$ be a predicate name with an arity n, let $f, f_1, \ldots, f_n \in \mathcal{F}$ be features, and let $u = f_1 \ldots f_n$ be a feature chain. Then the following expression is also a concept term:

$$\exists u_1 \ldots u_n.P \quad \text{(existential predicate restriction)}$$

If $a \in \mathcal{O}_A$ is an abstract individual name and $x, x_1, \ldots, x_n \in \mathcal{O}_D$ are concrete individual names, then an ABox may contain also the following assertional axioms:

$$f(a, x)$$

$$P(x_1, \ldots, x_n)$$

It should be emphasized, that it is not allowed to use specific elements of $\Delta^{\mathcal{D}}$ (e.g., numbers) in the assertions. E.g., it is not permitted to write $size(shoes, 36)$. However, it is possible to express the fact, that the shoe size is 36, with help of a unary predicate $=_{36}$ (interpreted as a singleton set $\{36\}$) and the following two assertions: $size(shoes, x)$ and $=_{36}(x)$. ∎

Definition 3.10 ($\mathcal{ALC}(\mathcal{D})$ syntax in terms of a stand-alone ontology formalism) In terms of an $\mathcal{F}$-ontology formalism, additionally to Σ_c, Σ_r and Σ_o discussed in Definition 3.5, an alphabet Σ of $\mathcal{ALC}(\mathcal{D})$ includes a set of *value-domain names* Σ_d (that is a set of names for concrete domains), a set of *value-relation names* Σ_u (to be interpreted as concrete predicate names, associated with an arity n), a set of *attribute-names* Σ_a (for feature names), a set of *value-constants* Σ_v (denotes elements of the concrete domain

$\Delta^{\mathcal{D}}$) and a set of *variable names* Σ_x, which we use for referring to concrete individuals in an ABox. The concept language $\mathcal{L}_C$ provides now for an additional construct for building concepts with features and predicates (of the form $\exists u_1 \ldots u_n.P$). The TBox language $\mathcal{L}_T$ contains all possible $\mathcal{ALC}(\mathcal{D})$ TBoxes as before, whereas the ABox language $\mathcal{L}_A$ consists of $\mathcal{ALC}(\mathcal{D})$ ABoxes with additional assertions as introduced in Definition 3.9. ∎

Definition 3.11 ($\mathcal{ALC}(\mathcal{D})$ semantics) An $\mathcal{ALC}(\mathcal{D})$-interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta^{\mathcal{D}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the abstract domain), a non-empty set $\Delta^{\mathcal{D}}$ (the domain of the concrete domain $\mathcal{D}$) and a function $\cdot^{\mathcal{I}}$. The interpretation of concrete domains and variables is separated from interpretation of concepts and abstract individuals. Therefore, the sets $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{D}}$ must be disjoint. The interpretation of the assertional language is extended by mapping every (abstract) individual name from $\mathcal{O}_A$ to a single element of $\Delta^{\mathcal{I}}$ and every (concrete) individual name from $\mathcal{O}_D$ to a single element from $\Delta^{\mathcal{D}}$.

The interpretation function maps each feature name f to a partial function $f^{\mathcal{I}}$ from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{D}}$. If $u = f_1 \ldots f_n$ is a feature chain, then $u^{\mathcal{I}}$ denotes the composition $f_1^{\mathcal{I}} \circ \ldots \circ f_n^{\mathcal{I}}$ of the partial functions $f_1^{\mathcal{I}}, \ldots, f_n^{\mathcal{I}}$. Given feature chains $u_1, \ldots, u_n$, the existential predicate restriction is interpreted as follows:

$$(\exists(u_1, \ldots, u_n).P)^{\mathcal{I}} = \\ \{a \in \Delta^{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta^{\mathcal{D}} : (a, x_1) \in u_1^{\mathcal{I}} \wedge \cdots \wedge (a, x_n) \in u_n^{\mathcal{I}} \wedge (x_1, \ldots, x_n) \in P^{\mathcal{D}}\}$$

An interpretation $\mathcal{I}$ satisfies assertional axioms

$$f(a, x) \text{ iff } f^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}$$

$$P(x_1, \ldots, x_n) \text{ iff } (x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}$$

∎

Definition 3.12 ($\mathcal{ALC}(\mathcal{D})$ semantics in terms of a stand-alone ontology formalism) Analogous to Definition 3.7 we say that every $\mathcal{ALC}(\mathcal{D})$ interpretation $\mathcal{I}$ corresponds to an FOL interpretation of an $\mathcal{F}$-ontology. For every concrete domain $\Delta^{\mathcal{D}}$, every concept name C, every role name R, every attribute name f, every predicate name P, every abstract object a and every concrete object x it holds that:

$$\begin{array}{lll} \Delta^{\mathcal{I}} = \Delta_o^{\mathcal{I}} & \Delta^{\mathcal{D}} \subseteq \Delta_v^{\mathcal{I}} & C^{\mathcal{I}} \subseteq \Delta_o^{\mathcal{I}} \\ R^{\mathcal{I}} \subseteq \Delta_o^{\mathcal{I}} \times \Delta_o^{\mathcal{I}} & f^{\mathcal{I}} \subseteq (\Delta_o^{\mathcal{I}})^n \times \Delta_v^{\mathcal{I}} & \\ a^{\mathcal{I}} \in \Delta_o^{\mathcal{I}} & x^{\mathcal{I}} \in \Delta_v^{\mathcal{I}} & \end{array}$$

∎

3.2.2 Transitive roles

In the previous subsection, functional roles (features) are introduced. Besides functionality, transitivity of roles is identified also as an important requirement in many practical applications. Integration of transitive roles into the language $\mathcal{ALC}$ led to the language $\mathcal{ALC}_{R^+}$ (for details see [Sat96]).

From the syntactic point of view, the set of all role names $\mathcal{R}$ is divided into disjoint sets of role names $\mathcal{S}$ and transitive role names $\mathcal{R}_+$. The semantics of transitive roles is, that for every transitive role $R \in \mathcal{R}_+$ the interpretation $\mathcal{I}$ must satisfy additionally the following restriction:

$$\text{if } (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \text{ and } (b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}} \text{ then } (a^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$$

If a DL with transitive roles is considered as an instantiation of an $\mathcal{F}$-ontology, then the concept language $\mathcal{L}_C$ have to be extended in an appropriate way.

3.2.3 Role hierarchies

Another important extension of DLs are role hierarchies ([HS99]). Using this feature it is possible to postulate that some relations are subrelations of other relations. Formally, a role inclusion axiom is defined as an expression of the form $R \sqsubseteq S$, where R and S are roles or inverse roles. A role hierarchy R_h is a finite set of role inclusion axioms. The relation $\sqsubseteq^\star$ is defined as reflexive and transitive closure of $\sqsubseteq$ over a role hierarchy R_h. Given $\sqsubseteq^\star$, a role R is called a sub-role of a role S if $R \sqsubseteq^\star S$. A super-role is defined analogously.

If a DL provides for role hierarchies, then besides GCIs the TBox contains also a finite set of role inclusion axioms. This affects the TBox language $\mathcal{L}_T$ in an appropriate manner.

3.2.4 Number restrictions

Number restrictions, that allow for an individual to restrict the number of its role fillers in a certain way, are available in almost all DL systems [HN90, HB91a, Sch94]. Putting these constructs into DLs was motivated in particular by technical applications. Number restrictions are concept construction operators of the form $(\leq n\, R)$ or $(\geq n\, R)$ (simple number restrictions) and $(\leq n\, R.C)$ or $(\geq n\, R.C)$ (qualified number restrictions [HB91b]). For simple number restrictions, interpretations must satisfy

$$\begin{aligned} (\leq n\, R)^{\mathcal{I}} &= \{x \mid \#\{y|(x,y) \in R^{\mathcal{I}}\} \leq n\} \\ (\geq n\, R)^{\mathcal{I}} &= \{x \mid \#\{y|(x,y) \in R^{\mathcal{I}}\} \geq n\} \end{aligned}$$

For qualified number restrictions, interpretations must satisfy

$$\begin{aligned} (\leq n\, R.C)^{\mathcal{I}} &= \{x \mid \#\{y|(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\} \\ (\geq n\, R.C)^{\mathcal{I}} &= \{x \mid \#\{y|(x,y) \in \textstyle\int R \wedge y \in \int C\} \geq n\} \end{aligned}$$

Due to requirements of decidability, in (qualified) number restrictions only simple roles are allowed. A role is called simple if it is neither transitive nor has transitive sub-roles (see [HST99] for details).

DLs with number restrictions can be instantiated in the proposed ontology framework in a similar manner as $\mathcal{ALC}$. Here, the concept language $\mathcal{L}_C$ is extended by appropriate concept building terms, which can consist of non-negative integers in a binary or unary notation.

3.3 Ontology services

In the proposed ontology formalism, an $\mathcal{F}$-ontology is equipped with a range of so-called *$\mathcal{F}$-ontology services* which can be exploited for application tasks. Among the common services, a distinguished class of so-called reasoning services is identified, which are independent of the syntactic representation of the ontology. For both classes of services, a syntactic level is defined in terms of a *signature*, a set of *preconditions* and a domain for the result.

Meaningful examples for reasoning services offered by DL systems are the following so-called standard inference problems:

Knowledge base satisfiability. This is the main inference service provided by DL-based reasoning systems. A model of the TBox is an interpretation which satisfies all axioms of a TBox. The ABox satisfiability problem (w.r.t. a TBox) is to check whether there exists an interpretation (a model of the TBox) that satisfies all ABox assertions. An interpretation which satisfies a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, is called a model of the knowledge base ($\mathcal{T}$, $\mathcal{A}$).

Concept satisfiability. A concept D subsumes a concept C w.r.t. a TBox $\mathcal{T}$ if for all models $\mathcal{I}$ of $\mathcal{T}$ it holds that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. D is called the subsumer, C is the subsumee. A concept name A_1 mentioned in a TBox is called a most-specific subsumer of a concept name A_2 (mentioned in the TBox and different from A_1) if A_1 subsumes A_2 and there is no other concept name A_3 (mentioned in the TBox and different from A_1 and A_2) such that A_1 subsumes A_3 and A_3 subsumes A_2. The least general subsumee of a concept name is defined analogously.

Classification of knowledge bases. The classification problem for a TBox is to find the set of most-specific subsumers of every concept name mentioned in the TBox (or knowledge base). The induced graph is called the subsumption hierarchy of the TBox.

Instance checking. The instance problem $instance_{(\mathcal{T},\mathcal{A})}(i, C)$ w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ is to test whether $i^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of the knowledge base. We say that $instance_{(\mathcal{T},\mathcal{A})}(i, C)$ is entailed.

Instance retrieval. The instance retrieval problem $retrieve_{(\mathcal{T},\mathcal{A})}(C)$ w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ and a query concept C is to determine all individuals i mentioned in the ABox for which $instance(i, C)$ is entailed.

NSI. Examples of so-called non-standard inference services are computing the least common subsumer (LCS), computing the most specific concept (MSC), concept matching, concept approximation etc.

Other DL services which depend on the syntactical form of an ontology (a knowledge base) are often related to querying the knowledge base structure on the meta-level, e.g., for obtaining a told information. For these services, the $\mathcal{F}$-ontology (TBox or ABox) must

be always specified as an argument. Examples of such meta services are the following retrieval problems, just to mention some of them:

- Get all concept names from the specified TBox.
- Return all individuals of the specified ABox.
- Get all concepts from the TBox, which subsume the specified concept.
- Get all roles from the TBox, that the given role subsumes.
- Get all concepts of which the individual is an instance.

3.3.1 DL inferences as instantiations of $\mathcal{F}$-ontology services

In order to illustrate how an $\mathcal{F}$-ontology service can be instantiated, we consider some DL-based inferences.

Instance checking. The inference problem *instance checking*, as discussed above, can be described as an $\mathcal{F}$-ontology service as follows:

- The signature specifies the name of the service, e.g., *instance_checking*.
- The signature further specifies that the service is called with 3 arguments: a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, an individual name $i \in \Sigma_o$, and a concept $C \in \mathcal{L}_C$.
- As precondition one can demand that i must be mentioned in the ABox $\mathcal{A}$.
- The domain of the result is $\{\mathsf{true}, \mathsf{false}\}$.

The service $instance\_checking(\mathcal{K}, i, C)$ returns true if $i^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of the knowledge base $\mathcal{K}$ and false otherwise.

Instance retrieval. The inference problem *instance retrieval*, as discussed above, can be described as an $\mathcal{F}$-ontology service as follows:

- The service name is, e.g., *instance_retrieval*.
- There are 2 arguments: a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a concept $C \in \mathcal{L}_C$.
- The domain of the result is 2^{Σ_o}.

The *meaning* of this service is to find all individuals i mentioned in the ABox $\mathcal{A}$ for which $instance\_checking(\mathcal{K}, i, C)$ is entailed.

Meta-querying service. Finally, consider the meta-querying service "get all concept names from the specified TBox" as an instantiation of an $\mathcal{F}$-ontology service. This service can be described as follows:

- The name of the service is, e.g., *all_atomic_concepts*.
- There is one argument: a TBox $\mathcal{T}$.
- The domain of the result is 2^{Σ_c}.

The service *all_atomic_concepts*($\mathcal{T}$) returns all concepts names mentioned in the TBox $\mathcal{T}$.

3.4 Remark on a standard DL as situated ontology

For practical DL applications, it is often useful to distinguish between assertions made in an ABox of a "core" ontology and descriptions of bulk data. Although these descriptions can be viewed theoretically just as a part of an ABox, to put they into a core ontology does not make much sense from the practical point of view. Bulk data descriptions can be stored, e.g., in external ABoxes and loaded on demand by a DL system. Considering this scenario, we talk about a standard DL ontology situated in some external system (so-called environment) which is a resource of bulk data.

4 The OWL 1.1 extension of the Web Ontology Language

In this section, we present the OWL 1.1 extension of the W3C Web Ontology Language as a particular instance of our ontology framework [2]. In Section 4.1 we introduce OWL 1.1 and discuss the main influences in its design. In Section 4.2, we present the syntax of OWL 1.1 and, finally, in Section 4.3 we represent OWL 1.1 as an ontology formalism.

4.1 Overview

The initial design of the OWL was quite conservative, and features that did not receive widespread support within the working group were excluded from the language. Features for which effective reasoning methods were not known (or expected to be shortly known) were also not included.

OWL 1.1 is a simple extension to OWL DL species of the W3C OWL Web Ontology Language. OWL 1.1 has been designed to provide some interesting and useful expressive additions to OWL DL while retaining the desirable characteristics of OWL DL, including decidability and effective implementability. In particular, OWL 1.1:

1. adds language features commonly requested by users of OWL DL;

[2] We refer the reader to `http://owl1-1.cs.manchester.ac.uk/` for more information on OWL 1.1.

2. is known to be decidable, and for which practical decision procedures have been designed; and

3. is likely to be implemented by the developers of OWL DL reasoners.

OWL 1.1 has borrowed heavily from recent research on Description Logics as well as from recent research on the nature of the Semantic Web.

4.1.1 OWL 1.1 and description logics

OWL DL is based on a description logic called $\mathcal{SHOIN}$. Even when OWL DL was designed, there were discussions as to whether it should be based on $\mathcal{SHOIN}$ or on $\mathcal{SHOIQ}$ [HS05], the latter being the former's extension with *qualifying number restrictions*. This expressive means is rather useful for modeling and is known to be no more problematical for a reasoner than the unqualified number restrictions in OWL DL. Interestingly, an effective decision procedure for $\mathcal{SHOIN}$ and $\mathcal{SHOIQ}$ has only been designed recently [HS05], and is already implemented in reasoners for OWL DL. Additionally, there have recently been two streams of work on extensions to the Description Logic underlying OWL DL. Firstly, there has been considerable work on how best to add datatypes and relationships between data values to OWL-like languages. The general ideas and requirements are basically similar—the datatypes themselves need to be effectively representable, and the relationships closed under boolean operations and computable—but the various proposals differ in detail.

Secondly, extensions to expressive description logics allowing more expressive property constructs have been devised and investigated. This line of work has lead to the $\mathcal{RIQ}$ [HS03], $\mathcal{SRIQ}$ [HKS05] and $\mathcal{SROIQ}$ [HKS06] description logics, and effective reasoning processes for them.

The existence of this work in the Description Logic community has made it simple to add qualified number restrictions, enhanced property constructs, and more expressive datatypes. OWL 1.1 essentially takes this work in entirely and without significant modification.

4.1.2 OWL 1.1 and OWL

As OWL 1.1 is a simple extension to OWL DL, it borrows heavily from OWL DL. To this end, OWL 1.1 uses the same basic syntax style as the "abstract" syntax for OWL DL [PSHH04]. As well as using the same syntactic style, OWL 1.1 incorporates the entire OWL DL syntax, only providing extensions to it. In this way, any legal OWL DL ontology is also a legal OWL 1.1 ontology.

As well, the meaning of OWL 1.1 is compatible with the meaning of OWL DL. Instead of providing a direct model-theoretic semantics, the meaning of OWL 1.1 is provided by a mapping to the Description Logic $\mathcal{SROIQ}$.

On the other hand, OWL 1.1 does not provide any significant features provided by OWL Full over OWL DL. This is largely because OWL 1.1 is essentially a Description Logic, and the facilities provided by OWL Full over OWL DL (meta-modelling, blending objects and datatypes, unusual syntactic forms, subverting basic constructs, etc.) are essentially those that go outside of the Description Logic paradigm.

Expressive ontology languages, such as OWL 1.1 and OWL DL, though decidable, have a high worst-case computational complexity[3] and are hard to use and implement efficiently. The design of simpler ontology languages with more tractable inferences was considered of primary importance by the W3C Web Ontology Working Group. The OWL Lite subset of OWL DL was designed as a language that is easier to use and present to naive users, as well as easier to implement.

The Web Ontology Working Group concluded that the main complexity of OWL DL relies on the possibility of defining complex boolean descriptions using, for example, union and complement; as a consequence, OWL Lite explicitly prohibits unions and complements in the definition of concepts; additionally, OWL Lite limits all descriptions in the scope of a quantifier to concept names, does not allow individuals to show up as concepts, and limits cardinalities to 0 and 1. The goal was to significantly reduce the number of available modeling constructs, on the one hand, and to eliminate the major sources of non-determinism in reasoning, on the other hand.

Although OWL Lite looks much simpler than OWL DL, it is still possible to express more complex concept descriptions by introducing new concept names, exploiting implicit negations and using axioms to associate multiple descriptions with a given concept name. So, from a user perspective, OWL Lite is even harder to use than OWL DL, since the available modeling constructs do not correspond to the actual expressivity of the language. Also, from a computational perspective, OWL Lite is only slightly less complex than OWL DL (namely ExpTime-complete instead of NExpTime-complete [Tob00]), and all the important reasoning problems remain intractable. In contrast to OWL, OWL 1.1 does not single out just one language subset. Instead, various subsets of OWL 1.1 have been identified, each of which benefits from tractable (i.e., polynomial time) reasoning for one or more important reasoning tasks. The intention is that these subsets can be used and implemented as appropriate to a particular application.

4.2 OWL 1.1 syntax

The "abstract" syntax of of OWL 1.1 is an extension of the "abstract" syntax for OWL DL. The extensions in OWL 1.1 lift its expressive power to that of $\mathcal{SROIQ}$ [HKS06]. This amounts to adding qualified cardinality restrictions, as an extension to restrictions on datatype properties and object properties; local reflexivity restrictions for simple properties, as an extension to restrictions on object properties; reflexive, irreflexive, and anti-symmetric flags for simple properties, as an extension to the flags allowed on object properties; and disjointness of simple and datatype properties, and *regular* property inclusion axioms, as new axioms.

dataRestrictionComponent ::= dataCardinality
dataCardinality ::= minCardinality(non-negative-integer dataRange)
| maxCardinality(non-negative-integer dataRange)
| cardinality(non-negative-integer dataRange)
individualRestrictionComponent ::= individualCardinality

[3]Satisfiability and subsumption are NExpTime-complete for $\mathcal{SHOIQ}$, and ExpTime-complete for $\mathcal{SHIQ}$.

individualCardinality ::= minCardinality(non-negative-integer description)
| maxCardinality(non-negative-integer description)
| cardinality(non-negative-integer description)
individualRestrictionComponent ::= self
individualvaluedPropertyFlags ::= Reflexive | Irreflexive
Symmetric | AntiSymmetric
axiom ::= DisjointProperties(datavaluedPropertyID)
| DisjointProperties(individualvaluedPropertyID)
axiom ::= SubPropertyOf(propertyChain(individualvaluedPropertyID)
individualvaluedPropertyID)

Only simple properties (i.e., properties that are neither transitive, nor have a transitive subProperty [HKS06]) can: have the self restriction component; be specified as being Reflexive, Irreflexive, Symmetric, or Antisymmetric; or be used in DisjointProperties axioms for individual-valued properties.

The SubPropertyOf axioms involving individual-valued properties must be *regular*. That is, there must be a strict partial order, $<$, on individual-valued properties such that for each SubPropertyOf axiom involving individual-valued properties, of the form SubPropertyOf(S R), S is the inverse of R, S is of the form propertyChain(R ... R), S is of the form propertyChain(S1 ... Sn) and each Si $<$ R, S is of the form propertyChain(R S1 ... Sn) and each Si $<$ R, or S is of the form propertyChain(S1 ... Sn R) and each Si $<$ R.

The first couple of other additions in OWL 1.1 are simple syntactic sugar. To make the common construct of multiple disjoint classes easier to state, OWL 1.1 provides an axiom that directly states that a group of classes are pairwise disjoint, instead of having to use separate disjoint axioms for each pair of classes. To make the common construct of stating that an individual does not have a particular property value, OWL 1.1 provides a construct directly for this, instead of, e.g., having to state that the individual is an instance of a suitable restriction class.

axiom ::= DisjointUnion(description)
value ::= valueNot(individualvaluedPropertyID individualID)
| valueNot(individualvaluedPropertyID individual)
| valueNot(datavaluedPropertyID dataLiteral)

OWL 1.1 includes its own methods for user-defined datatypes, using a syntax similar to the one used in Protégé. The semantics for OWL 1.1 user-defined datatypes is taken from XML Schema Datatypes [BM01].

dataRange ::= datatype(datatypeID { datatypeRestriction })
datatypeRestriction ::= datatypeFacet(dataLiteral)
datatypeFacet ::= length | minLength | maxLength | pattern | enumeration
| maxInclusive | maxExclusive | minInclusive minExclusive
| totalDigits | fractionDigits
axiom ::= Datatype(datatypeID { annotation } base(datatypeID)
{ datatypeRestriction })

Datatype facets should only by used where they would be allowed in XML Schema Datatypes, except that the length, minLength, maxLength, and pattern facets are not allowed for numeric types. Datatype facets have the same meaning as in XML Schema Datatypes, except that they uniformly work in the value space, never the lexical space. If a datatype facet is used in a way that has no meaning, such as (length 5∧∧xsd:string), then the datatype extension is empty.

OWL 1.1 allows restrictions that relate values for different data-valued properties on the same individual.

restriction ::= holds(datatypePredicateID { argument })
restriction ::= datatypePropertyID dataLiteral
datatypePredicateID ::= equal | notEqual | lessThan | lessThanEqual
| greaterThan | greaterThanEqual

The syntax here allows an arbitrary number of arguments, but must be appropriate for the predicate, and all the current predicates only allow two arguments. All invalid combinations are unsatisfiable (i.e., they do not signal an error). The equality and order for a particular base type is taken from XML Schema Datatypes. If a base datatype does not have an order then the ordering restriction is unsatisfied.

OWL 1.1 removes the limitation imposed in OWL DL that the sets of class, individual and property names must be pairwise disjoint. The semantic change to allow this without computational consequences is to break the RDF-inspired connection between class and property extensions and the individual denotation of names. With this change, any name can be made the subject of a non-annotation property, but in this (syntactic) context the name is always (semantically) interpreted as an individual.

4.3 OWL 1.1 as an instantiation of the TONES framework for stand-alone ontologies

In this section, we define the ontology formalism $\mathcal{F}_{OWL}$ for stand-alone ontologies and provide a mapping σ between the OWL 1.1 syntax described in the previous section into the concept, TBox, and ABox languages of $\mathcal{F}_{OWL}$. The semantics of $\mathcal{F}_{OWL}$ as given in this section, together with the translation σ specify the semantics of OWL 1.1. The semantics given here is equivalent to the one given in the OWL 1.1 specification [4].

We consider a finite or countably infinite alphabet Σ, partitioned into alphabets Σ_c of *concept names*, Σ_r of *relation names* with arity 2, Σ_d of *value-domain names*, Σ_u of *value-relation names*, Σ_a of *attribute names*, Σ_o of *object-constants*, Σ_v of *value-constants*.

Since OWL 1.1 allows *punning* in the signature, we will assume that the sets $\Sigma_c \cup \Sigma_d$, $\Sigma_r \cup \Sigma_a$ and Σ_o are not pair-wise disjoint and thus the same name can be used in a knowledge base to denote a concept, a relation or an attribute and an object-constant, contrary to what was required in OWL-DL. This simple form of meta-modelling *does not* provide additional expressive power, nor involves semantic consequences. The disjointness of Σ_r and Σ_a and of Σ_c and Σ_d is still enforced. Finally, the set Σ_v of value-constants is kept disjoint with all other sets in the vocabulary. We assume that Σ_d is given by

[4] http://owl1-1.cs.manchester.ac.uk/Semantics.html.

the set of *supported datatypes*; in OWL 1.1., the set of supported datatypes is the set of user-defined datatypes that are allowed in the XML Schema Datatype specification.

The concept language $\mathcal{L}_C$ is given by the following grammar rules, which specify how concepts and relations are constructed respectively:

$$\begin{array}{rcl} C & ::= & A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \{o\} \\ & & \mid \exists R.D \mid \forall R.D \mid \geq nS.C \mid \leq nS.C \mid \exists R.\mathsf{Self} \\ & & \mid \exists U.\Phi \mid \forall U.\Phi \mid \geq nU.\Phi \mid \leq nU.\Phi \mid \exists U_1 \cdots U_n.p_n \end{array}$$

$$R ::= R^-$$

In the grammar rules above, $A \in \Sigma_c$, C_i, D are concepts, R, S are relations, with S a *simple* relation [5] n is a non-negative integer, $o \in \Sigma_o$, $\Phi \in \Sigma_d$, $U \in \Sigma_a$, and p_n an n-ary relation in Σ_u.

Table 1 specifies the semantics for the concept language $\mathcal{L}_C$. In the table, we use the symbol # to denote the cardinality of a set.

The TBox language $\mathcal{L}_T$ is specified by the following grammar:

$$\begin{array}{rcl} \alpha & ::= & \mathsf{Trans(R)} \mid \mathsf{ASymm(R)} \mid \mathsf{Ref(R)} \mid \mathsf{Irr(R)} \\ & & \mid S_1 \cdots S_n \sqsubseteq R \mid \mathsf{Dis(R,S)} \mid \mathsf{U} \sqsubseteq \mathsf{V} \mid \mathsf{Dis(U,V)} \\ & & \mid C \sqsubseteq D \end{array}$$

with $R, S_{(i)} \in \Sigma_r$, $U, V \in \Sigma_a$ and C, D concepts. Furthermore, all the relations in sentences of the for $\mathsf{Irr(R)}$, ASymmR or $\mathsf{Dis(R,S)}$ must be simple.

The ABox language $\mathcal{L}_A$ is given by the following grammar:

$$\begin{array}{rcl} \alpha & ::= & C(a) \mid a \doteq b \mid a \dot{\neq} b \mid R(a,b) \mid \neg R(a,b) \\ & & \mid U(a,\phi) \mid \neg U(a,\phi) \end{array}$$

with $a, b \in \Sigma_o$, $R \in \Sigma_r$, $U \in \Sigma_a$ and $\phi \in \Sigma_v$.

Table 2 specifies the semantics of the TBox language $\mathcal{L}_T$ and the ABox language $\mathcal{L}_A$. These grammars and tables completely specify the ontology formalism $\mathcal{F}_{OWL}$ associated to OWL 1.1.

The specification of OWL 1.1 in terms of the ontology formalism $\mathcal{F}_{OWL}$ is completed with the translation from an OWL 1.1 abstract syntax as described in the previous section into the concept, TBox and ABox languages we have just defined.

We define a *translation* function that maps OWL 1.1 ontologies in abstract syntax into equivalent ontologies in our ontology formalism. The translation function and the semantics of the ontology formalism completely specify the semantics of OWL 1.1.

[5] We refer interested reader to [HKS06] for the definition of a simple relation.

Conmplex Concepts	Syntax And Semantics
Conjunction	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Negation	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} - C^{\mathcal{I}}$
Existential Restrict.	$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.\langle a,b\rangle \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$
Universal Restrict.	$(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall b.\langle a,b\rangle \in R^{\mathcal{I}}, \text{ implies } b \in C^{\mathcal{I}}\}$
Nominals	$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$
Qualified	$(\geq nR.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \#(\{b \mid \langle a,b\rangle \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}) \geq n\}$
Number Restrict.	$(\geq nR.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \#(\{b \mid (a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}) \leq n\}$
Local Reflexivity	$(\exists R.\mathsf{Self})^{\mathcal{I}} = \{\mathsf{a} \in \Delta^{\mathcal{I}} \mid \langle \mathsf{a},\mathsf{a}\rangle \in \mathsf{R}^{\mathcal{I}}\}$
Concrete Existential	$(\exists U.\Phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists \phi.(a,\phi) \in U^{\mathcal{I}} \text{ and } \phi \in \Phi^{\mathcal{D}}\}$
Concrete Universal	$(\forall U.\Phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall \phi.(a,\phi) \in U^{\mathcal{I}}, \text{ implies } \phi \in \Phi^{\mathcal{D}}\}$
Concrete	$(\geq nU.\Phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \#(\{\phi \mid (a,\phi) \in U^{\mathcal{I}} \text{ and } \phi \in \Phi^{\mathcal{D}}\}) \geq n\}$
Number Restrict.	$(\leq nU.\Phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \#(\{\phi \mid (a,\phi) \in U^{\mathcal{I}} \text{ and } \phi \in \Phi^{\mathcal{D}}\}) \leq n\}$
n-ary Existential	$(\exists U_1 \ldots U_n.p_n)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists \phi_1 \ldots \phi_n.\langle a,\phi_1\rangle \in U_1^{\mathcal{I}}, \ldots, \langle a,\phi_n\rangle \in U_n^{\mathcal{I}}$ and $\langle \phi_1 \ldots \phi_n\rangle \in p_n^{\mathcal{D}}\}$
Complex Roles	
Role Inversion	$(R^-)^{\mathcal{I}} = \{\langle a,b\rangle \mid \langle b,a\rangle \in R^{\mathcal{I}}\}$

Table 1: Concept language and Semantics

TBox Language	Syntax And Semantics
Transitivity	$\mathcal{I} \models \mathsf{Trans(R)}$ if $\langle a,b\rangle$ and $\langle b,c\rangle \in R^{\mathcal{I}}$ implies $\langle a,c\rangle \in R^{\mathcal{I}}$
Antisimmetry	$\mathcal{I} \models \mathsf{ASymm(R)}$ if $\langle a,b\rangle \in R^{\mathcal{I}}$ and $\langle b,a\rangle \in R^{\mathcal{I}}$ implies $a = b$
Reflexivity	$\mathcal{I} \models \mathsf{Ref(R)}$ if $\langle a,b\rangle \in R^{\mathcal{I}}$ implies $\langle a,a\rangle \in R^{\mathcal{I}}$
Irreflexivity	$\mathcal{I} \models \mathsf{Irr(R)}$ if $\langle a,b\rangle \in R^{\mathcal{I}}$ implies $a \neq b$
Role Inclusion	$\mathcal{I} \models S_1 \cdots S_n \sqsubseteq R$ if $S_1^{\mathcal{I}} \circ \cdots \circ S_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
Role Disjointness	$\mathcal{I} \models \mathsf{Dis(R,S)}$ if $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
Concept Inclusion	$\mathcal{I} \models C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concrete Role Inclusion	$\mathcal{I} \models U \sqsubseteq V$ if $U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$
Disjoint Concrete Roles	$\mathcal{I} \models \mathsf{Dis(U,V)}$ if $U^{\mathcal{I}} \cap V^{\mathcal{I}} = \emptyset$
ABox Language	Syntax and Semantics
Concept Assertion	$\mathcal{I} \models C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$
Individual equality	$\mathcal{I} \models a \doteq b$ if $a^{\mathcal{I}} = b^{\mathcal{I}}$
Individual inequality	$\mathcal{I} \models a \dot{\neq} b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$
Role Assertion	$\mathcal{I} \models R(a,b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$
Negated Role Asser.	$\mathcal{I} \models \neg R(a,b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \notin R^{\mathcal{I}}$
Concrete Assertion	$\mathcal{I} \models U(a,\phi)$ if $\langle a^{\mathcal{I}}, \phi^{\mathcal{D}}\rangle \in U^{\mathcal{I}}$
Neg. Concrete Assertion	$\mathcal{I} \models \neg U(a,\phi)$ if $\langle a^{\mathcal{I}}, \phi^{\mathcal{D}}) \notin U^{\mathcal{I}}$

Table 2: TBox Language, ABox Language and their Semantics

Let $\mathcal{O}_0$ be an OWL 1.1 ontology in abstract syntax and let $\{\mathcal{O}_j\}_{1\leq j\leq m}$ be the set of ontologies imported (directly or indirectly) by $\mathcal{O}_0$; let $\{\alpha_1^j, \ldots, \alpha_{n_j}^j\}$ for $0 \leq j \leq m$ be the set of axioms and facts contained in $\mathcal{O}_j$. The translation into a $\mathcal{F}_{OWL}$ ontology $\mathcal{O}'$ is as follows: $\mathcal{O}' = \bigcup_{0\leq j\leq m} \sigma(\mathcal{O}_j)$, where $\sigma(\mathcal{O}_j) = \bigcup_{1\leq i\leq n_j} \sigma(\alpha_i^j)$. names occurring in $\mathcal{O}'$. The syntactic correspondence between OWL 1.1 descriptions and the concept language $\mathcal{L}_C$ is given in Table 3 as is the correspondence between OWL 1.1 axioms and facts and the TBox and ABox languages respectively. This table completely specifies the translation function σ and should be read as follows: given a construct in OWL 1.1 abstract syntax in the first column, its evaluation under σ is given in the second column.

5 The Semantic Web Rules Language (SWRL)

In this section, we present the Semantic Web Rules Language (SWRL) [HPSBT05] as a particular instance of the TONES framework for stand-alone ontologies. In Section 5.1 we introduce the basics of SWRL. In Section 5.2, we present the syntax of SWRL and, finally, in Section 5.3 we represent SWRL as an ontology formalism.

5.1 Overview

SWRL extends OWL DL [PSHH04] with a form of rules while maintaining maximum backwards compatibility with OWL's existing syntax and semantics. For such a purpose, SWRL adds a new kind of axiom to OWL DL, namely Horn clause rules, extending the OWL abstract syntax and the direct model-theoretic semantics for OWL DL to provide a formal semantics and syntax for OWL ontologies including such rules. SWRL is considerably more powerful than either OWL DL or Horn rules alone.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The informal meaning of a rule can be read as: whenever (and however) the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Both the antecedent (body) and consequent (head) of a rule consist of zero or more atoms. Atoms can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL DL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values. Atoms are satisfied in extended interpretations (to take care of variables) in the usual model-theoretic way, i.e., the extended interpretation maps the variables to domain elements in a way that satisfies the description, property, sameAs, or differentFrom, just as in the regular OWL model theory.

Multiple atoms in an antecedent are treated as a conjunction. An empty antecedent is thus treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation.

Multiple atoms in a consequent are treated as separate consequences, i.e., they must all be satisfied. In keeping with the usual treatment in rules, an empty consequent is treated as trivially false (i.e., not satisfied by any extended interpretation). Such rules are satisfied if and only if the antecedent is not satisfied by any extended interpretation. Note that rules with multiple atoms in the consequent could easily be rewritten (by applying

OWL 1.1 Abstract Syntax	$\mathcal{F}_{OWL}$ Syntax
A (Class URI)	Concept name A
`owl:Thing`	$\top$
`owl:Nothing`	$\bot$
R (Object Property URI)	Relation $R \in \Sigma_r$
U (Datatype Prop. URI)	Attribute $U \in \Sigma_a$
a (Individual URI)	Object-constant $a \in \Sigma_o$
ϕ (Data Value or Plain Literal)	Value-constant $\phi \in \Sigma_v$
p_m (Datatype Predicate ID)	Value relation $p_m \in \Sigma_u$
Φ (OWL 1.1 Datatype)	Value-domain $\Phi \in \Sigma_d$
$\mathtt{intersectionOf}(C_1 \cdots C_n)$	$\sigma(C_1) \sqcap \cdots \sqcap \sigma(C_n)$
$\mathtt{unionOf}(C_1 \cdots C_n)$	$\sigma(C_1) \sqcup \cdots \sqcup \sigma(C_n)$
$\mathtt{complementOf}(C)$	$\neg\sigma(C)$
$\mathtt{oneOf}(a_1 \ \cdots \ a_n)$	$\{\sigma(a_1)\} \sqcup \cdots \sqcup \{\sigma(a_n)\}$
$\mathtt{oneOf}(\phi_1 \cdots \phi_n)$	$\{\sigma(\phi_1), \ldots, \sigma(\phi_n)\}$
$\mathtt{restriction}(R\ X_1 \cdots X_n)$	$\sigma(\mathtt{restriction}(R\ X_1)) \sqcap \cdots \sqcap \sigma(\mathtt{restriction}(R\ X_n))$
$\mathtt{restriction}(R\ \mathtt{someValuesFrom}(C))$	$\exists\sigma(R).\sigma(C)$
$\mathtt{restriction}(R\ \mathtt{allValuesFrom}(C))$	$\forall\sigma(R).\sigma(C)$
$\mathtt{restriction}(R\ \mathtt{hasValue}(a))$	$\exists\sigma(R).\{\sigma(a)\}$
$\mathtt{restriction}(R\ \mathtt{self})$	$\exists\sigma(R).\mathsf{Self}$
$\mathtt{restriction}(R\ \mathtt{minCardinality}(n))$	$\geq n\sigma(R).\top$
$\mathtt{restriction}(R\ \mathtt{maxCardinality}(n))$	$\leq n\sigma(R).\top$
$\mathtt{restriction}(R\ \mathtt{Cardinality}(n))$	$= n\sigma(R).\top$
$\mathtt{restriction}(R\ \mathtt{minCardinality}(n\ C))$	$\geq n\sigma(R).\sigma(C)$
$\mathtt{restriction}(R\ \mathtt{maxCardinality}(n\ C))$	$\leq n\sigma(R).\sigma(C)$
$\mathtt{restriction}(R\ \mathtt{Cardinality}(n\ C))$	$= n\sigma(R).\sigma(C)$
$\mathtt{restriction}(U\ X_1 \cdots X_n)$	$\sigma(\mathtt{restriction}(U\ X_1)) \sqcap \cdots \sqcap \sigma(\mathtt{restriction}(U\ X_n))$
$\mathtt{restriction}(U\ \mathtt{someValuesFrom}(\Phi))$	$\exists\sigma(U).\sigma(\Phi)$
$\mathtt{restriction}(U\ \mathtt{allValuesFrom}(\Phi))$	$\forall\sigma(U).\sigma(\Phi)$
$\mathtt{restriction}(U\ \mathtt{hasValue}(\phi))$	$\exists\sigma(U).\{\sigma(\phi)\}$
$\mathtt{restriction}(U\ \mathtt{minCardinality}(n))$	$\geq n\sigma(U)$
$\mathtt{restriction}(U\ \mathtt{maxCardinality}(n))$	$\leq n\sigma(U)$
$\mathtt{restriction}(U\ \mathtt{Cardinality}(n))$	$= n\sigma(U)$
$\mathtt{restriction}(U\ \mathtt{minCardinality}(n\ \Phi))$	$\geq n\sigma(U).\sigma(\Phi)$
$\mathtt{restriction}(U\ \mathtt{maxCardinality}(n\ \Phi))$	$\leq n\sigma(U).\sigma(\Phi)$
$\mathtt{restriction}(U\ \mathtt{Cardinality}(n\ \Phi))$	$= n\sigma(U).\sigma(\Phi)$
$\mathtt{restriction}(\text{holds}(p_m)\ U_1 \cdots U_m)$	$\exists\sigma(U_1) \cdots \sigma(U_m).\sigma(p_m)$

OWL 1.1 Abstract Syntax	$\mathcal{F}_{OWL}$ Syntax
$\mathtt{ObjectProperty}(R\ \mathtt{super}(R_1) \cdots \mathtt{super}(R_m)$	$\bigcup_{i=1}^{m}\{\sigma(R) \sqsubseteq \sigma(R_i)\}$
$\mathtt{domain}(C_1) \cdots \mathtt{domain}(C_m)$	$\bigcup_{i=1}^{m}\{\geq 1\sigma(R) \sqsubseteq \sigma(C_i)\}$
$\mathtt{range}(C_1) \cdots \mathtt{range}(C_m)$	$\bigcup_{i=1}^{m}\{\top \sqsubseteq \forall\sigma(R).\sigma(C_i)\}$
$\mathtt{inverseOf}(S)$	$\{\sigma(R) \equiv \sigma(S)^-\}$
[`Symmetric`]	$\{\sigma(R) \equiv \sigma(R)^-\}$
[`Functional`]	$\{\top \sqsubseteq \leq 1\sigma(R)\}$
[`InverseFunctional`]	$\{\top \sqsubseteq \leq 1\sigma(R)^-\}$
[`Transitive`]	$\{\mathsf{Trans}(\sigma(\mathsf{R}))\}$
[`Reflexive`]	$\{\mathsf{Ref}(\sigma(\mathsf{R}))\}$
[`Irreflexive`]	$\{\mathsf{Irr}(\sigma(\mathsf{R}))\}$
[`AntiSymmetric`])	$\{\mathsf{ASymm}(\sigma(\mathsf{R}))\}$
$\mathtt{SubPropertyOf}(R_1\ R_2)$	$\{\sigma(R_1) \sqsubseteq \sigma(R_2)\}$
$\mathtt{SubPropertyOf}(\mathtt{propertyChain}(R_1 \cdots R_m)\ R)$	$\{\sigma(R_1) \cdots \sigma(R_m) \sqsubseteq \sigma(R)\}$
$\mathtt{DisjointProperties}(R_1\ R_2)$	$\{\mathsf{Dis}(\sigma(\mathsf{R}_1), \sigma(\mathsf{R}_2))\}$
$\mathtt{EquivalentProperties}(R_1 \cdots R_m)$	$\bigcup_{i=1}^{m-1}\{\sigma(R_i) \equiv \sigma(R_{i+1})\}$
$\mathtt{DataProperty}(U\ \mathtt{super}(U_1) \cdots \mathtt{super}(U_m)$	$\bigcup_{i=1}^{m}\{\sigma(U) \sqsubseteq \sigma(U_i)\}$
$\mathtt{domain}(C_1) \cdots \mathtt{domain}(C_m)$	$\bigcup_{i=1}^{m}\{\geq 1\sigma(U) \sqsubseteq \sigma(C_i)\}$
$\mathtt{range}(\Phi_1) \cdots \mathtt{range}(\Phi_m)$	$\bigcup_{i=1}^{m}\{\top \sqsubseteq \forall\sigma(U).\sigma(\Phi_i)\}$
[`Functional`])	$\{\top \sqsubseteq \leq 1\sigma(U)\}$
$\mathtt{SubPropertyOf}(U_1 U_2)$	$\{\sigma(U_1) \sqsubseteq \sigma(U_2)\}$
$\mathtt{EquivalentProperties}(U_1 \cdots U_m)$	$\bigcup_{i=1}^{m-1}\{\sigma(U_i) \equiv \sigma(U_{i+1})\}$
$\mathtt{DisjointProperties}(U_1\ U_2)$	$\{\mathsf{Dis}(\sigma(\mathsf{U}_1), \sigma(\mathsf{U}_2))\}$
$\mathtt{Class}(A\ \mathtt{partial}\ C_1 \cdots C_m)$	$\{\sigma(A) \sqsubseteq \sigma(C_1) \sqcap \cdots \sqcap \sigma(C_m)\}$
$\mathtt{Class}(A\ \mathtt{complete}\ C_1 \cdots C_m)$	$\{\sigma(A) \equiv \sigma(C_1) \sqcup \cdots \sqcup \sigma(C_m)\}$
$\mathtt{EnumeratedClass}(A\ a_1 \cdots a_m)$	$\{\sigma(A) \equiv \{\sigma(a_1)\} \sqcup \cdots \sqcup \{\sigma(a_m)\}\}$
$\mathtt{EquivalentClasses}(C_1 \cdots C_m)$	$\bigcup_{i=1}^{m-1}\{\sigma(C_i) \equiv \sigma(C_{i+1})\}$
$\mathtt{DisjointClasses}(C_1 \cdots C_m)$	$\bigcup_{i,j=1;i\neq j}^{m}\{\sigma(C_i) \sqsubseteq \neg\sigma(C_j)\}$
$\mathtt{DisjointUnion}(C\ C_1 \cdots C_m)$	$\{\sigma(C) \equiv \sigma(C_1) \sqcup \cdots \sqcup \sigma(C_m)\} \cup \bigcup_{i,j=1;i\neq j}^{m}\{\sigma(C_i) \sqsubseteq \neg\sigma(C_j)\}$
$\mathtt{Individual}(a\ \mathtt{type}(C_1) \cdots \text{type}(C_m)$	$\bigcup_{i=1}^{m}\{\sigma(C_i)(\sigma(a))\}$
$\mathtt{value}(R_1\ \ b_1) \cdots \mathtt{value}(R_m\ \ b_m)$	$\bigcup_{i=1}^{m}\{\sigma(R_i)(\sigma(a), \sigma(b_i))\}$
$\mathtt{value}(U_1\ \phi_1) \cdots \mathtt{value}(U_m\ \phi_m)$	$\bigcup_{i=1}^{m}\{\sigma(U_i)(\sigma(a), \sigma(\phi_i))\}$
$\mathtt{valueNot}(R_1\ b_1) \cdots \mathtt{valueNot}(R_m\ b_m)$	$\bigcup_{i=1}^{n}\{\neg\sigma(R_i)(\sigma(a), \sigma(b_i))\}$
$\mathtt{valueNot}(U_1\ \phi_1) \cdots \mathtt{valueNot}(U_m\ \phi_m)\ )$	$\bigcup_{i=1}^{m}\{\neg\sigma(U_i)(\sigma(a), \sigma(\phi_i))\}$
$\mathtt{SameIndividual}(a_1 \cdots a_m)$	$\bigcup_{i,j=1;i\neq j}^{m}\{\sigma(a_i) \doteq \sigma(a_j)\}$
$\mathtt{DifferentIndividuals}(a_1 \cdots a_m)$	$\bigcup_{i,j=1;i\neq j}^{m}\{\sigma(a_i) \dot{\neq} \sigma(a_j)\}$

Table 3: Translation of OWL 1.1 abstract syntax into $\mathcal{F}_{OWL}$

standard rules of distributivity) into multiple rules each with an atomic consequent.

OWL DL becomes undecidable when extended in this way as rules can be used to simulate role value maps [SS89] and make it easy to encode known undecidable problems as a SWRL ontology consistency problem.

Adding rules to description logic based knowledge representation languages is far from being a new idea. Several early description logic systems, e.g., Classic [PSMB+91, BPS94], included a rule language component. In these systems, however, rules were given a weaker semantic treatment than axioms asserting sub- and super-class relationships; they were only applied to individuals, and did not affect class based inferences such as the computation of the class hierarchy. More recently, the CARIN system integrated rules with a description logic in such a way that sound and complete reasoning was still possible [LR98]. This could only be achieved, however, by using a rather weak description logic (*much* weaker than OWL), and by placing severe syntactic restrictions on the occurrence of description logic terms in the (heads of) rules. Similarly, the DLP language proposed in [GHVD03] is based on the intersection of a description logic with horn clause rules; the result is obviously a decidable language, but one that is necessarily less expressive than either the description logic or rules language from which it is formed.

SWRL illustrates how a simple form of Horn-style rules can be added to the OWL language in a syntactically and semantically coherent manner.

5.2 SWRL syntax

The syntax for SWRL in this section abstracts from any exchange syntax for OWL and thus facilitates access to and evaluation of the language. This syntax extends the abstract syntax of OWL described in the OWL Semantics and Abstract Syntax document.

Like the OWL abstract syntax, we will specify the abstract syntax for rules by means of a version of Extended BNF, very similar to the Extended BNF notation used for XML. In this notation, terminals are quoted; non-terminals are not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Whitespace is ignored in the productions given here.

Names in the abstract syntax are RDF URI references.

The meaning of each construct in the abstract syntax for rules is informally described when it is introduced. The formal meaning of these constructs is given in the SWRL specification [HPSBT05], and is also specified, coherently with the one in the SWRL specification, in Section 5.3, when we define SWRL as an instantiation of the TONES framework for stand-alone ontologies.

From the OWL Semantics and Abstract Syntax document, an OWL ontology in the abstract syntax contains a sequence of annotations, axioms, and facts. Axioms may be of various kinds, for example, subClass axioms and equivalentClass axioms. This proposal extends axioms to also allow rule axioms, by adding the production:

axiom ::= rule

Thus a SWRL ontology could contain a mixture of rules and other OWL DL constructs, including ontology annotations, axioms about classes and properties, and facts about

OWL individuals, as well as the rules themselves.

A rule axiom consists of an antecedent (body) and a consequent (head), each of which consists of a (possibly empty) set of atoms. Just as for class and property axioms, rule axioms can also have annotations. These annotations can be used for several purposes, including giving a label to the rule by using the rdfs:label annotation property.

rule ::= 'Implies('{annotation} antecedent consequent')'
antecedent ::= 'Antecedent('{atom}')'
consequent ::= 'Consequent('{atom}')'

Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold. An empty antecedent is treated as trivially holding (true), and an empty consequent is treated as trivially not holding (false). Non-empty antecedents and consequents hold iff all of their constituent atoms hold. As mentioned above, rules with multiple consequents could easily be rewritten (using standard rules of distributivity) into multiple rules each with a single atomic consequent.

Atoms in rules can be of the form C(x), P(x,y), Q(x,z), sameAs(x,y) or differentFrom(x,y), where C is an OWL DL description, P is an OWL DL *individual-valued* Property, Q is an OWL DL *data-valued* Property, x,y are either variables or OWL individuals, and z is either a variable or an OWL data value. In the context of OWL Lite, descriptions in atoms of the form C(x) may be restricted to class names.

atom ::= description '(' i-object ')'
| individualvaluedPropertyID '(' i-object i-object ')'
| datavaluedPropertyID '(' i-object d-object ')'
| sameAs '(' i-object i-object ')'
| differentFrom '(' i-object i-object ')'

Informally, an atom C(x) holds if x is an instance of the class description C, an atom P(x,y) (resp. Q(x,z)) holds if x is related to y (z) by property P (Q), an atom sameAs(x,y) holds if x is interpreted as the same object as y, and an atom differentFrom(x,y) holds if x and y are interpreted as different objects.

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule. As usual, only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as "safety").

i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral

i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'

5.3 SWRL as an instantiation of the TONES framework

In this section, we define the ontology formalism $\mathcal{F}_{SWRL}$ and provide a mapping the SWRL syntax described in the previous section into the concept, TBox and ABox languages of $\mathcal{F}_{SWRL}$. The semantics of $\mathcal{F}_{SWRL}$ as given in this section, together with the mapping

specify the semantics of SWRL. The semantics given here is equivalent to the one given in the SWRL specification [HPSBT05].

In SWRL, we consider a finite or countably infinite alphabet Σ, partitioned into alphabets Σ_c of *concept names*, Σ_r of *relation names* with arity 2, Σ_d of *value-domain names*, Σ_u of *value-relation names*, Σ_a of *attribute names*, Σ_o of *object-constants*, Σ_v of *value-constants*.

As in OWL 1.1, we assume that Σ_d is given by the set of of *supported datatypes*. The set of supported datatypes is the set of user-defined datatypes that are allowed in the XML Schema Datatype specification.

The concept language $\mathcal{L}_C$ is exactly the same as the one defined for OWL 1.1 and its semantics is given in the very same way.

In order to define the TBox and ABox languages, we define first the set of *atoms*. We assume disjoint sets V_o and V_d of object and data *variables*, which are also disjoint with Σ, and define the atoms using the following grammar:

$$\mathbf{a} \;::=\; C(x) \mid R(x,y) \mid U(x,v) \mid x \doteq y \mid x \dot{\neq} y$$

with $x, y \in V_o \cup \Sigma_o$, C a concept in $\mathcal{L}_C$, $R \in \Sigma_r$, $U \in \Sigma_a$ and $v \in V_v \cup \Sigma_v$. An atom is *ground* if it does not contain variables.

The TBox language $\mathcal{L}_T$ is specified by the following grammar:

$$\begin{aligned} \alpha \;::=\; & \mathsf{Trans(R)} \mid \mathsf{S} \sqsubseteq \mathsf{R} \mid \mathsf{U} \sqsubseteq \mathsf{V} \mid \mathsf{C} \sqsubseteq \mathsf{D} \\ & \mid \mathbf{a}_1 \cdots \mathbf{a}_m \leftarrow \mathbf{b_1}, \cdots, \mathbf{b_n} \end{aligned}$$

with $R, S_{(i)} \in \Sigma_r$, $U, V \in \Sigma_a$ and C, D concepts, and $\mathbf{a_{(i)}}, \mathbf{b_{(i)}}$ are atoms. Furthermore, we assume that the rule $\mathbf{a}_1 \cdots \mathbf{a}_m \leftarrow \mathbf{b_1}, \cdots, \mathbf{b_n}$ is safe, as given in the previous section, and not ground. The atoms $\mathbf{a}_1 \cdots \mathbf{a}_m$ constitute the *consequent* of the rule and the atoms $\mathbf{b_1}, \cdots, \mathbf{b_n}$ are the *antecedent*; we assume that the antecedent may be empty.

The ABox language $\mathcal{L}_A$ is given by the following grammar:

$$\begin{aligned} \alpha \;::=\; & C(a) \mid a \doteq b \mid a \dot{\neq} b \mid R(a,b) \mid U(a,\phi) \\ & \mid \mathbf{f_1} \cdots \mathbf{f_m} \leftarrow \mathbf{g_1}, \cdots, \mathbf{g_n} \end{aligned}$$

with $a, b \in \Sigma_o$, $R \in \Sigma_r$, $U \in \Sigma_a$ and $\phi \in \Sigma_v$ and and $\mathbf{f_{(i)}}, \mathbf{g_{(i)}}$ ground atoms

All the sentences in $\mathcal{L}_T$ and $\mathcal{L}_A$ except for the rules are identical as the ones in OWL 1.1 and their semantics is given in the same way. In order to specify the ontology formalism $\mathcal{F}_{SWRL}$ associated to SWRL, it only remains to specify the semantics of atoms and rules. A binding $B(\mathcal{I})$ extends an interpretation $\mathcal{I}$ by additionally mapping each object variable x to an element of $\Delta_o^{\mathcal{I}}$ and each value variable v to an element of $\Delta_v^{\mathcal{I}}$.

An atom is satisfied by a binding $B(\mathcal{I})$ under the conditions given in Table 4, where C is a concept, $R \in \Sigma_r$, $U \in \Sigma_a$, x, y are object variables or object constants, and z is a value variable or a value constant. Given a rule:

$$\mathbf{a}_1 \cdots \mathbf{a}_m \leftarrow \mathbf{b_1}, \cdots, \mathbf{b_n}$$

Atom	Condition on Interpretation
$C(x)$	$x^{\mathcal{I}} \in C^{\mathcal{I}}$
$R(x, y)$	$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
$U(x, z)$	$\langle x^{\mathcal{I}}, z^{\mathcal{I}} \rangle \in U^{\mathcal{I}}$
$x \dot{=} y$	$x^{\mathcal{I}} = y^{\mathcal{I}}$
$x \dot{\neq} y$	$x^{\mathcal{I}} \neq y^{\mathcal{I}}$

Table 4: Interpretation Conditions

A binding $B(\mathcal{I})$ satisfies the antecedent $\mathbf{b_1} \cdots \mathbf{b_n}$ iff $n = 0$ (i.e the antecedent is empty) or $B(\mathcal{I})$ satisfies every atom $\mathbf{b_{(i)}}$. A binding $B(\mathcal{I})$ satisfies the consequent $\mathbf{a_1}, \ldots, \mathbf{a_n}$ iff $B(\mathcal{I})$ satisfies every atom $\mathbf{a}_i$. A rule is satisfied by an interpretation $\mathcal{I}$ iff for every binding B such that $B(\mathcal{I})$ satisfies the antecedent, $B(\mathcal{I})$ also satisfies the consequent.

The syntactic correspondence between SWRL and $\mathcal{F}_{SWRL}$ is obtained by simply extending the translation function σ used for OWL 1.1 in the straightforward way to deal with atoms and rules.

6 Linking data to ontologies: The description logic *DL-Lite*$_A$

In several areas (e.g., Enterprise Application Integration, Data Integration [Len02], and the Semantic Web [HH01]) the intensional level of the application domain can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system. One of the most interesting usages of such shared conceptualizations is *ontology-based data access*. That is, to the usual data layer of an information system we superimpose a conceptual layer to be exported to the client. Such a layer allows the client to have a conceptual view of the information in the system, which abstracts away from how such information is maintained in the data layer of the system itself. While ontologies are the best candidates for realizing the conceptual layer, relational DBMSs are natural candidates for the management of the data layer, since relational database technology is nowadays the best technology for efficient management of very large quantities of data.

Recently, basic research has been done in understanding which fragments of OWL, OWL-DL, or OWL-Lite would be suited to act as the formalism for representing ontologies in this context [CDGL$^+$06a, OCE06, HMS05]. The outcome of this work is that none of the variants of OWL is suitable, if not restricted (they all are coNP-hard w.r.t. data complexity). Possible restrictions that guarantee polynomial reasoning (at least, if we concentrate on instance checking only) have been looked at, such as Horn-$\mathcal{SHIQ}$ [HMS05], $\mathcal{EL}^{++}$ [BBL05], DLP [GHVD03]. Among such fragments, of particular interest are those belonging to the DL-Lite family [CDGL$^+$05, CDGL$^+$06a]. These logics allow for answering complex queries (namely, conjunctive queries, i.e., SQL select-project-join queries, and unions of conjunctive queries) in LogSpace w.r.t. data complexity (i.e., the complexity measured only w.r.t. the size of the data). More importantly, they allow for delegating

query processing, after a preprocessing phase which is independent of the data, to the relational DBMS managing the data layer.

According to [CDGL$^+$06a] there are two maximal languages in the DL-Lite family that possess the above property. The first one is *DL-Lite*$_F$, which allows for specifying the main modeling features of conceptual models, including cyclic assertions, ISA on concepts, inverses on roles, role typing, mandatory participation to roles, and functional restrictions on roles. The second one is *DL-Lite*$_R$, which is able to fully capture (the DL fragment of) RDFS, and has in addition the ability of specifying mandatory participation on roles and disjointness between concepts and roles. The language obtained by unrestrictedly merging the features of *DL-Lite*$_F$ and *DL-Lite*$_R$, while quite interesting in general, is not in LogSpace w.r.t. data complexity anymore [CDGL$^+$06a], and hence loses the most interesting computational feature for ontology-based data access.

In this paper, we look in more detail at the interaction between the distinguishing features of *DL-Lite*$_F$ and *DL-Lite*$_R$, and we single out an extension of both *DL-Lite*$_F$ and *DL-Lite*$_R$ that is still LogSpace w.r.t. data complexity, and allows for delegating the data dependent part of the query answering process to to the relational DBMS managing the data layer.

Moreover, we take seriously the distinction in OWL between objects and values (a distinction that typically is blurred in description logics), and introduce, besides concepts and roles, also concept-attributes and role-attributes, that describe properties of concepts (resp., roles) represented by values rather than objects. In fact, role attributes are currently not available in OWL, but are present in most conceptual modeling formalisms such as UML class diagrams and Entity-Relationship diagrams.

We look at the problem of accessing databases that are independent from the ontology, and are related to the ontology through suitable mappings [Len02]. Observe, however, that such databases, being relational, store only values (not objects), and hence objects need to be constructed from such values, i.e., we have to deal with the so-called "impedance mismatch".

Finally, we show how the description logics described in this section, called *DL-Lite*$_{FR}$ and *DL-Lite*$_A$, can be regarded as an instance of the ontology framework.

6.1 The description logic *DL-Lite*$_{FR}$

In this section we present a new logic of the *DL-Lite* family, called *DL-Lite*$_{FR}$. As usual in DLs, all logics of the *DL-Lite* family allow one to represent the universe of discourse in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. In addition, the DLs discussed in this paper allow one to use (*i*) value-domains, a.k.a. concrete domains [BH91], denoting sets of (data) values, (*ii*) concept attributes, denoting binary relations between objects and values, and (*iii*) role attributes, denoting binary relations between pairs of objects and values. Obviously, a role attribute can also be seen as a ternary relation relating two objects and a value.

6.1.1 Syntax

We now introduce the DL *DL-Lite*$_{FR}$, that combines the main features of two DLs presented in [CDGL$^+$06a], called *DL-Lite*$_F$ and *DL-Lite*$_R$, respectively, and forms the basics of *DL-Lite*$_A$, that will be defined later in Section 6.3. In providing the specification of our logics, we use the following notation.

Definition 6.1 (*DL-Lite*$_{FR}$ notation)

- A denotes an *atomic concept*, B a *basic concept*, and C a *general concept*;
- D denotes an *atomic value-domain*, E a *basic value-domain*, and F a *general value-domain*;
- P denotes an *atomic role*, Q a *basic role*, and R a *general role*;
- U_C denotes an *atomic concept attribute*, and V_C a *general concept attribute*;
- U_R denotes an *atomic role attribute*, and V_R a *general role attribute*;
- $\top_C$ denotes the *universal concept*, $\top_D$ denotes the *universal value-domain*.

■

Given a concept attribute U_C (resp. a role attribute U_R), we call the *domain* of U_C (resp. U_R), denoted by $\delta(U_C)$ (resp. $\delta(U_R)$), the set of objects (resp. of pairs of objects) that U_C (resp. U_R) relates to values, and we call *range* of U_C (resp. U_R), denoted by $\rho(U_C)$ (resp. $\rho(U_R)$), the set of values that U_C (resp. U_R) relates to objects (resp. pairs of objects). Notice that the domain $\delta(U_C)$ of a concept attribute U_C is a concept, whereas the domain $\delta(U_R)$ of a role attribute U_R is a role. Furthermore, we denote with $\delta_F(U_C)$ (resp. $\delta_F(U_R)$) the set of objects (resp. of pairs of objects) that U_C (resp. U_R) relates to values in the value-domain F.

Definition 6.2 *DL-Lite*$_{FR}$ expressions are defined inductively as follows.

- Concept expressions:

$$\begin{array}{rcl} B & ::= & A \mid \exists Q \mid \delta(U_C) \\ C & ::= & \top_C \mid B \mid \neg B \mid \exists Q.C \mid \delta_F(U_C) \mid \exists \delta_F(U_R) \mid \exists \delta_F(U_R)^- \end{array}$$

- Value-domain expressions (*rdfDataType* denotes predefined value-domains such as integers, strings, etc.):

$$\begin{array}{rcl} E & ::= & D \mid \rho(U_C) \mid \rho(U_R) \\ F & ::= & \top_D \mid E \mid \neg E \mid \mathit{rdfDataType} \end{array}$$

- Attribute expressions:

$$\begin{array}{rcl} V_C & ::= & U_C \mid \neg U_C \\ V_R & ::= & U_R \mid \neg U_R \end{array}$$

- Role expressions:

$$\begin{array}{rcl} Q & ::= & P \mid P^- \mid \delta(U_R) \mid \delta(U_R)^- \\ R & ::= & Q \mid \neg Q \mid \delta_F(U_R) \mid \delta_F(U_R)^- \end{array}$$

■

A *DL-Lite$_{FR}$ knowledge base* (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is constituted by two components: a TBox $\mathcal{T}$, used to represent intensional knowledge, and an ABox $\mathcal{A}$, used to represent extensional knowledge.

Definition 6.3 A *DL-Lite$_{FR}$ TBox* is constituted by a set of assertions of the form:

$B \sqsubseteq C$	*concept inclusion assertion*
$Q \sqsubseteq R$	*role inclusion assertion*
$E \sqsubseteq F$	*value-domain inclusion assertion*
$U_C \sqsubseteq V_C$	*concept attribute inclusion assertion*
$U_R \sqsubseteq V_R$	*role attribute inclusion assertion*
(funct P)	*role functionality assertion*
(funct P^-)	*inverse role functionality assertion*
(funct U_C)	*concept attribute functionality assertion*
(funct U_R)	*role attribute functionality assertion*

■

A concept inclusion assertion expresses that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions. A role functionality assertion expresses the (global) functionality of an atomic role. Analogously for the other types of functionality assertions.

As for the ABox, we introduce two disjoint alphabets, called Γ_O and Γ_V, respectively. Symbols in Γ_O, called *object-constants*, are used to denote objects, while symbols in Γ_V, called *value-constants*, are used to denote data values.

Definition 6.4 A *DL-Lite$_{FR}$* ABox is a finite set of assertions of the form:

$$A(a), \quad D(c), \quad P(a,b), \quad U_C(a,c), \quad U_R(a,b,c) \qquad \textit{membership assertions}$$

where a and b are constants in Γ_O, and c is a constant in Γ_V. ■

6.1.2 Semantics

The semantics of *DL-Lite$_{FR}$* is given in terms of FOL interpretations.

Definition 6.5 An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a first order structure over the interpretation domain $\Delta^{\mathcal{I}}$ that is the disjoint union of $\Delta^{\mathcal{I}}_O$ and $\Delta^{\mathcal{I}}_V$, with an *interpretation function* $\cdot^{\mathcal{I}}$ such that

- for all $a \in \Gamma_O$, we have that $a^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$;
- for all $a, b \in \Gamma_O$, we have that $a \neq b$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$;
- for all $c \in \Gamma_V$, we have that $c^{\mathcal{I}} \in \Delta_V^{\mathcal{I}}$;
- for all $c, d \in \Gamma_V$, we have that $c \neq d$ implies $c^{\mathcal{I}} \neq d^{\mathcal{I}}$;
- and the following conditions are satisfied (below, $o, o' \in \Delta_O^{\mathcal{I}}$, and $v \in \Delta_V^{\mathcal{I}}$; moreover, we consider $\delta(U_C)$ and $\delta(U_R)$ as abbreviations for $\delta_{\top_D}(U_C)$ and $\delta_{\top_D}(U_R)$, respectively):

$\top_C^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$
$\top_D^{\mathcal{I}} = \Delta_V^{\mathcal{I}}$
$A^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}$
$D^{\mathcal{I}} \subseteq \Delta_V^{\mathcal{I}}$
$P^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$
$U_C^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$
$U_R^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$
$(\neg U_C)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U_C^{\mathcal{I}}$
$(\neg U_R)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U_R^{\mathcal{I}}$
$(\rho(U_C))^{\mathcal{I}} = \{\ v \mid \exists o.\, (o, v) \in U_C^{\mathcal{I}}\ \}$
$(\rho(U_R))^{\mathcal{I}} = \{\ v \mid \exists o, o'.\, (o, o', v) \in U_R^{\mathcal{I}}\ \}$

$(P^-)^{\mathcal{I}} = \{\ (o, o') \mid (o', o) \in P^{\mathcal{I}}\ \}$
$(\delta_F(U_C))^{\mathcal{I}} = \{\ o \mid \exists v.\, (o, v) \in U_C^{\mathcal{I}} \wedge v \in F^{\mathcal{I}}\ \}$
$(\delta_F(U_R))^{\mathcal{I}} = \{\ (o, o') \mid \exists v.\, (o, o', v) \in U_R^{\mathcal{I}} \wedge v \in F^{\mathcal{I}}\ \}$
$(\delta_F(U_R)^-)^{\mathcal{I}} = \{\ (o, o') \mid \exists v.\, (o', o, v) \in U_R^{\mathcal{I}} \wedge v \in F^{\mathcal{I}}\ \}$
$(\exists \delta_F(U_R))^{\mathcal{I}} = \{\ o \mid \exists\ o'.\, (o, o') \in (\delta_F(U_R))^{\mathcal{I}}\ \}$
$(\exists \delta_F(U_R)^-)^{\mathcal{I}} = \{\ o \mid \exists\ o'.\, (o, o') \in (\delta_F(U_R)^-)^{\mathcal{I}}\ \}$
$(\exists Q)^{\mathcal{I}} = \{\ o \mid \exists o'.\, (o, o') \in Q^{\mathcal{I}}\ \}$
$(\exists Q.C)^{\mathcal{I}} = \{\ o \mid \exists o'.\, (o, o') \in Q^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\ \}$
$(\neg Q)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$
$(\neg B)^{\mathcal{I}} = \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}}$
$(\neg E)^{\mathcal{I}} = \Delta_V^{\mathcal{I}} \setminus E^{\mathcal{I}}$

■

We define when an interpretation $\mathcal{I}$ satisfies an assertion (i.e., is a model of the assertion) as follows (below, each e, possibly with subscript, is an element of either $\Delta_O^{\mathcal{I}}$ or $\Delta_V^{\mathcal{I}}$, depending on the context, each t, possibly with subscript, is a constant of either Γ_O or Γ_V, depending on the context, a and b are constants in Γ_O, and c is a constant in Γ_V).

Definition 6.6 An interpretation $\mathcal{I}$ *satisfies*:

- an inclusion assertion $\alpha \sqsubseteq \beta$, if $\alpha^{\mathcal{I}} \subseteq \beta^{\mathcal{I}}$;
- a functional assertion (funct γ), where γ is either P, P^-, or U_C, if, for each e_1, e_2, e_3, $(e_1, e_2) \in \gamma^{\mathcal{I}}$ and $(e_1, e_3) \in \gamma^{\mathcal{I}}$ implies $e_2 = e_3$;
- a functional assertion (funct U_R), if for each e_1, e_2, e_3, e_4, $(e_1, e_2, e_3) \in U_R^{\mathcal{I}}$ and $(e_1, e_2, e_4) \in U_R^{\mathcal{I}}$ implies $e_3 = e_4$;
- a membership assertion $\alpha(t)$, where α is either A or D, if $t^{\mathcal{I}} \in \alpha^{\mathcal{I}}$;
- a membership assertion $\beta(t_1, t_2)$, where β is either P or U_C, if $(t_1^{\mathcal{I}}, t_2^{\mathcal{I}}) \in \beta^{\mathcal{I}}$;
- a membership assertion $U_R(a, b, c)$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}, c^{\mathcal{I}}) \in U_R^{\mathcal{I}}$.

A *model of a KB* $\mathcal{K}$ is an interpretation $\mathcal{I}$ that is a model of all assertions in $\mathcal{K}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* an assertion α if all models of $\mathcal{K}$ are also models of α. ■

6.2 *DL-Lite*$_{FR}$ as an instantiation of the TONES framework for stand-alone ontologies

We define now the ontology formalism $\mathcal{F}_{\textit{DL-Lite}_{FR}} = \langle \Sigma, \mathcal{L}_C, \mathcal{L}_T, \mathcal{L}_A, \mathit{Sem} \rangle$ for stand-alone ontologies that corresponds to *DL-Lite*$_{FR}$. Its components are defined as follows.

- The alphabet Σ, is partitioned into the following alphabets:
  - Σ_c of *concept names*, i.e., the alphabet of atomic concepts (denoted by A in Definition 6.1) including the universal concept $\top_C$;
  - Σ_r of *relation names*, i.e., the alphabet of atomic roles (denoted by P in Definition 6.1);
  - Σ_d of *value-domain names*, i.e., the alphabet of atomic value-domains (denoted by D in Definition 6.1), including the universal value-domain $\top_D$;
  - Σ_u of *value-relation names*, which is empty, since *DL-Lite*$_A$ does not include explicit relations between domains;
  - Σ_a of *attribute names*, which is bipartite into two sub-alphabets of atomic concept attributes (denoted by U_C in Definition 6.1) and atomic role attributes (denoted by U_R in Definition 6.1);
  - Σ_o of *object-constants* is the alphabet Γ_O used to denote objects;
  - Σ_v of *value-constants* is the alphabet Γ_V used to denote data values.
- The concept language $\mathcal{L}_C$ is the language for expression of concepts, value-domains, attributes, and roles, introduced in Definition 6.2.
- The TBox language $\mathcal{L}_T$ is the language of TBox as introduced in Definition 6.3.
- The ABox language $\mathcal{L}_A$ is the language of ABox as introduced in Definition 6.4.
- The semantic specification is a triple $(\mathcal{S}, \delta, \circ)$, where:
  - $\mathcal{S}$ is the set of all FOL interpretations for Σ.
  - δ is a function, that, given a FOL interpretation $\mathcal{I} \in \mathcal{S}$ and a TBox or an ABox $\alpha \in \mathcal{L}_T \cup \mathcal{L}_A$, returns true if all assertions in α are satisfied according to the semantics of Definition 6.6, false otherwise.
  - $\circ$ is set intersection.

Some of the fundamental services for *DL-Lite*$_{FR}$ are, among others, knowledge base satisfiability, concept satisfiability, classification of knowledge bases, instance checking, and query answering. It has been shown in [CDGL$^+$06c] that all such services can be reformulated in terms of query answering. We therefore illustrate the instantiation of *DL-Lite*$_{FR}$ services to the framework by showing how the query answering service is instantiated. In order to do that, we first need to introduce a query formalism for *DL-Lite*$_{FR}$.

A conjunctive query (CQ) q over a knowledge base $\mathcal{K}$ is an expression of the form $q(\vec{x}) \leftarrow \exists \vec{y}.\mathit{conj}(\vec{x}, \vec{y})$, where $\vec{x}$ are the so-called *distinguished variables*, $\vec{y}$ are existentially

quantified variables called the *non-distinguished* variables, and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $A(x_o)$, $P(x_o, y_o)$, $D(x_v)$, $U_C(x_o, x_v)$, or $U_R(x_o, y_o, x_v)$, where x_o, y_o are either variables in $\vec{x}$ and $\vec{y}$ (called *object variables*) or constants in Γ_O, whereas x_v is either a variable in $\vec{x}$ and $\vec{y}$ (called a *value variable*) or a constant in Γ_V. A *union of conjunctive queries* (UCQ) is a query of the form $q(\vec{x}) \leftarrow \bigvee_i \exists \vec{y_i}.conj(\vec{x}, \vec{y_i})$

A query $q(\vec{x}) \leftarrow \varphi(\vec{x})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{e} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that, when we assign $\vec{e}$ to the variables $\vec{x}$, the formula $\varphi(\vec{x})$ evaluates to true in $\mathcal{I}$.

In terms of the framework for stand-alone ontologies, the *query answering* service has, at the syntactic level:

- A signature specifying a name for the service, e.g., *certain*, and the fact that there are two arguments: a knowledge base $\mathcal{K}$ and a query (CQ or UCQ) $q(\vec{x})$ over $\mathcal{K}$.
- A precondition imposing that the knowledge base $\mathcal{K}$ be satisfiable.
- The domain for the result, defined as $\{\mathsf{true}, \mathsf{false}\} \cup 2^{\mathcal{D}}$, where $\mathcal{D}$ is the set of all tuples of elements of $\Gamma_V \cup \Gamma_O$.

The meaning of this service is to return the *certain answers* to $q(\vec{x})$ over $\mathcal{K}$, i.e., all tuples $\vec{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that, when substituted to the variables $\vec{x}$ in $q(\vec{x})$, we have that $\mathcal{K} \models q(\vec{t})$, i.e., such that $\vec{t^{\mathcal{I}}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

6.3 Realizing services in *DL-Lite*$_{FR}$ and *DL-Lite*$_A$

As mentioned, it has been shown that services for description logics of the *DL-Lite* family can be reformulated in terms of query answering [CDGL$^+$06c]. We therefore concentrate now on the realization of such services.

6.3.1 From *DL-Lite*$_{FR}$ to *DL-Lite*$_A$

From the results in [CDGL$^+$06a] it follows that, in general, query answering over *DL-Lite*$_{FR}$ KBs is PTIME-hard in data complexity (i.e., the complexity measured w.r.t. the size of the ABox only). As a consequence, to solve query answering over *DL-Lite*$_{FR}$ KBs, we need at least the power of general recursive Datalog. Since we are interested in DLs where query ansering can be done in LOGSPACE, we now introduce the DL *DL-Lite*$_A$, which differs from *DL-Lite*$_{FR}$ because it imposes a limitation on the use of the functionality assertions in the TBox. As we will discuss, such limitation is sufficient to guarantee that query answering is in LOGSPACE w.r.t. data complexity, and thus it can be reduced to first-order query evaluation.

Definition 6.7 A *DL-Lite*$_A$ knowledge base is pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A}$ is a *DL-Lite*$_{FR}$ ABox, and $\mathcal{T}$ is a *DL-Lite*$_{FR}$ TBox satisfying the following conditions:

1. for every atomic or inverse of an atomic role Q appearing in a concept of the form $\exists Q.C$, the assertions $(\mathsf{funct}\ Q)$ and $(\mathsf{funct}\ Q^-)$ are not in $\mathcal{T}$;
2. for every role inclusion assertion $Q \sqsubseteq R$ in $\mathcal{T}$, where R is an atomic role or the inverse of an atomic role, the assertions $(\mathsf{funct}\ R)$ and $(\mathsf{funct}\ R^-)$ are not in $\mathcal{T}$;

3. for every concept attribute inclusion assertion $U_C \sqsubseteq V_C$ in $\mathcal{T}$, where V_C is an atomic concept attribute, the assertion (funct V_C) is not in $\mathcal{T}$;

4. for every role attribute inclusion assertion $U_R \sqsubseteq V_R$ in $\mathcal{T}$, where V_R is an atomic role attribute, the assertion (funct V_R) is not in $\mathcal{T}$.

■

Roughly speaking, a *DL-Lite*$_A$ TBox imposes the condition that *every functional role cannot be specialized* by using it in the right-hand side of role inclusion assertions; the same condition is also imposed on every functional (role or concept) attribute. It can be shown that functionalities specified in a *DL-Lite*$_A$ TBox are not implicitly propagated in the TBox, and that this guarantees membership in LogSpace for query answering.

We also observe here that the instantiation of the framework for stand-alone ontologies to *DL-Lite*$_A$ is the same as the one to *DL-Lite*$_{FR}$, with the only difference that the TBox language $\mathcal{L}_T$ must also satisfy the extra conditions imposed by Definition 6.7.

We now show a simple example of a *DL-Lite*$_{FR}$ TBox, with the aim of highlighting the use of attributes (in particular, role attributes). In the following, concept names are written capitalized, role names are written in uppercase, attribute names are written in lowercase, and domain names are in Courier font.

Example 6.8 Let $\mathcal{T}$ be the TBox containing the following assertions:

$$\begin{aligned}
\mathit{TempEmp} &\sqsubseteq \mathit{Employee} && (1)\\
\mathit{Manager} &\sqsubseteq \mathit{Employee} && (2)\\
\mathit{Employee} &\sqsubseteq \exists \mathit{WORKS\text{-}FOR}.\mathit{Project} && (3)\\
\delta(\mathit{until}) &\sqsubseteq \mathit{WORKS\text{-}FOR} && (4)\\
(\mathsf{funct}\ \mathit{until}) & && (5)\\
\mathit{TempEmp} &\sqsubseteq \exists \delta(\mathit{until}) && (6)\\
\mathit{Manager} &\sqsubseteq \exists \mathit{MANAGES} && (7)\\
\mathit{MANAGES} &\sqsubseteq \mathit{WORKS\text{-}FOR} && (8)\\
\mathit{Manager} &\sqsubseteq \neg\exists \delta(\mathit{until}) && (9)\\
\rho(\mathit{until}) &\sqsubseteq \mathtt{date} && (10)
\end{aligned}$$

The above TBox $\mathcal{T}$ models information about employees and projects. The assertions in $\mathcal{T}$ state that: (1) every temporary employee is an employee; (2) every manager is an employee; (3) every employee works for at least one project; (4) the domain of the role attribute *until* is the role *WORKS-FOR*; (5) the role attribute *until* is functional; (6) every temporary employee *must* participate in a role having an associated role attribute *until* (such a role, by assertion (4), is the role *WORKS-FOR*); (7) every manager participates to the role *MANAGES*; (8) every instance of role *MANAGES* is an instance of the role *WORKS-FOR*; (9) no manager *can* participate in a role having an associated attribute *until*; (10) the range of the role attribute *until* is `date`, i.e., every value associated by the attribute *until* to a role instance must be a value of type `date`. ■

6.3.2 Query answering in *DL-Lite*$_A$

We discuss now reasoning in *DL-Lite*$_A$, and concentrate on the basic reasoning task in the context of using ontologies to access large data repositories, namely (conjunctive) query answering over a *DL-Lite*$_A$ knowledge base. The other forms of reasoning usual in DLs can be reduced to query answering (through reductions analogous to the ones reported in [CDGL$^+$05] for *DL-Lite*).

We now briefly sketch the technique for query answering which we have defined for *DL-Lite*$_A$. In a nutshell, the algorithm has a structure similar to the methods developed for the logics in the *DL-Lite* family [CDGL$^+$05]: in particular, query answering is basically reduced to evaluation of a first-order query over a relational database representing the ABox. Such first-order query is obtained by reformulating the original query based on the TBox assertions. For *DL-Lite*$_A$, this reformulation can be obtained through the query reformulation technique for *DLR-Lite*$_R$ defined in [CDGL$^+$06a]. *DLR-Lite*$_R$ is a logic of the *DL-Lite* family which allows for expressing n-ary relations (but no functionality assertions). The possibility of reusing the query reformulation algorithm of *DLR-Lite*$_R$ is based on the fact that inclusion assertions in *DL-Lite*$_A$ can actually be expressed in *DLR-Lite*$_R$, as shown at steps 1 and 2 of the algorithm below.

An important aspect which such a query answering strategy relies on is a *separation* property between different kinds of TBox assertions: in particular, TBox assertions are classified into: (*i*) *positive inclusion assertions (PIs)*, i.e., inclusions having a positive concept/role/concept attribute/role attribute on its right-hand side, (*ii*) *negative inclusion assertions (NIs)*, i.e., inclusions having a negated concept/role/concept attribute/role attribute on its right-hand side; (*iii*) *functionality assertions*, i.e., assertions of the form (funct φ), where φ is a role/inverse of a role/atomic concept attribute/atomic role attribute. Then, it can be shown that, after saturating the TBox, as described at step 3 in the algorithm below, query answering can be done by considering in a separate way the set of PIs and the set of NIs and functionality assertions. More precisely, NIs and functionality assertions are relevant for the satisfiability of $\mathcal{K}$ (while PIs are not); moreover, in a satisfiable KB $\mathcal{K}$, only PIs are relevant for answering UCQs. We remark that the above separation property holds for *DL-Lite*$_A$, but it does not hold for *DL-Lite*$_{FR}$.

Let us now briefly describe the query answering algorithm. Given a *DL-Lite*$_A$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a Boolean union of conjunctive queries q, our algorithm proceeds as follows:

1. First, all qualified existential quantifications in the right-hand side of both concept and role inclusion assertions are compiled away by rewriting them through the use of auxiliary roles. More specifically, we modify the TBox $\mathcal{T}$ as follows:

   - the concept inclusion assertion $B \sqsubseteq \exists R.C$ is replaced by the inclusion assertions

     $$\begin{array}{ll} B \sqsubseteq \exists R_{aux} & \text{(concept inclusion assertion)} \\ \exists R_{aux}^- \sqsubseteq C & \text{(concept inclusion assertion)} \\ R_{aux} \sqsubseteq R & \text{(role inclusion assertion)} \end{array}$$

     where R_{aux} is a new role name.

- analogously, the role inclusion assertion $Q \sqsubseteq \delta_F(U_R)$ is replaced by the inclusion assertions

$$\begin{array}{ll} Q \sqsubseteq \delta(U_{aux}) & \text{(role inclusion assertion)} \\ \rho(U_{aux}) \sqsubseteq F & \text{(value-domain inclusion assertion)} \\ U_{aux} \sqsubseteq U_R & \text{(role attribute inclusion assertion)} \end{array}$$

  where U_{aux} is a new role attribute name.

Similar encodings allow us to compile away concept inclusion assertions of the form $B \sqsubseteq C$, where $C = \delta_F(U_C)$ or $C = \exists\delta_F(U_R)$ or $C = \exists\delta_F(U_R)^-$), and role attribute inclusion assertions of the form $Q \sqsubseteq \delta_F(U_R)^-$.

2. We convert the inclusion assertions in the TBox $\mathcal{T}$ into inclusion assertions in *DLR-Lite*$_R$. The transformation is based on considering each concept attribute as a binary relation, and each role attribute as a ternary relation, and on the following correspondences between the *DL-Lite*$_A$ notation and the *DLR-Lite*$_R$ notation:

$$\begin{array}{rcl} \delta(U_C) & \Rightarrow & U_C[1] \\ \rho(U_C) & \Rightarrow & U_C[2] \end{array} \qquad \begin{array}{rcl} \delta(U_R) & \Rightarrow & U_R[1,2] \\ \delta(U_R)^- & \Rightarrow & U_R[2,1] \\ \rho(U_R) & \Rightarrow & U_R[3] \end{array} \qquad \begin{array}{rcl} \exists\delta(U_R) & \Rightarrow & U_R[1] \\ \exists\delta(U_R)^- & \Rightarrow & U_R[2] \end{array}$$

3. Then, the TBox $\mathcal{T}$ is *saturated* by adding to $\mathcal{T}$ all negative inclusions logically implied by $\mathcal{T}$. This saturation step corresponds to the saturation of a *DLR-Lite*$_R$ knowledge base described in [CDGL$^+$06a].

4. We now verify satisfiability of $\langle \mathcal{T}, \mathcal{A} \rangle$. To do so, we simply check each functionality assertion and each NI in $\mathcal{A}$, in the following way: (i) for each functionality assertion evaluate a Boolean conjunctive query with inequality over $\mathcal{A}$, considering $\mathcal{A}$ as a relational database. Such a query is false in $\mathcal{A}$ iff the functionality assertion is not contradicted by the assertions in $\mathcal{A}$; (ii) for each NI evaluate a Boolean conjunctive query over $\mathcal{A}$, considering $\mathcal{A}$ as a relational database. Such a query is false in $\mathcal{A}$ iff the NI assertion is not contradicted by the assertions in $\mathcal{A}$. Details on such a satisfiability check can be found in [CDGL$^+$05].

5. If the above satisfiability check does not succeed, then the algorithm returns true (since there are no models for $\mathcal{K}$, every query is true in $\mathcal{K}$); otherwise, the query q is taken into account, and it is reformulated based on the TBox $\mathcal{T}$, according to the query reformulation procedure for *DLR-Lite*$_R$ described in [CDGL$^+$06a]. Let q' be the query obtained by this reformulation step.

6. Finally, the query q' is evaluated over $\mathcal{A}$, considering $\mathcal{A}$ as a relational database. The algorithm returns true iff q' is evaluated to true in $\mathcal{A}$.

The above algorithm can be used to decide the recognition problem associated with query answering in *DL-Lite*$_A$, i.e., establish whether a tuple $\vec{t}$ of constants is a certain answer to a UCQ q in a *DL-Lite*$_A$ KB $\mathcal{K}$ (the input to the algorithm is constituted by the Boolean UCQ $q(\vec{t})$ and the KB $\mathcal{K}$). Moreover, the algorithm can be immediately

generalized to query answering, i.e., to compute the set of all certain answers to a (non-Boolean) UCQ. Observe that steps 1–4 above are independent of the actual query q, and can be carried out on the knowledge base only.

Based on the above algorithm, we are able to provide an upper bound for the data complexity of answering UCQs in *DL-Lite*$_A$.

Theorem 6.9 *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_A$ KB. Answering UCQs posed to $\mathcal{K}$ is in* LogSpace *with respect to data complexity.*

We point out that the assumption that the TBox $\mathcal{T}$ is expressed in *DL-Lite*$_A$ rather than in *DL-Lite*$_{FR}$ is essential for the above upper bound to hold: in fact, from the results in [CDGL$^+$06a], it follows that, answering UCQs in *DL-Lite*$_{FR}$ is PTime-hard with respect to data complexity. This implies that the above query answering strategy cannot be pursued anymore, since in general there exists no FOL reformulation q' of a UCQ q (only depending on $\mathcal{T}$) such that the certain answers to q in $\mathcal{K}$ correspond to the evaluation of q' in $\mathcal{A}$.

6.4 Linking data to *DL-Lite*$_A$ ontologies

Most works on DLs do not deal with the problem of how to store ABox assertions, nor do they address the issue of how to acquire ABox assertions from existing data sources. It is our opinion that this topic is of special importance in several contexts where the use of ontologies is advocated, especially in the case where the ontology is used to provide a unified conceptual model of an organization (e.g., in Enterprise Application Integration). In these contexts, the problem can be described as follows: the ontology is a virtual representation of a universe of discourse, and the instances of concepts and roles in the ontology are simply an abstract representation of some real data stored in existing data sources. Therefore, the problem arises of establishing sound mechanisms for linking existing data to the instances of the concepts and the roles in the ontology.

In this section we sketch our solution to this problem, by presenting a mapping mechanism that enables a designer to link data sources to an ontology expressed in *DL-Lite*$_A$.

Before delving into the details of the method, a preliminary discussion on the notorious impedence mismatch problem between data and objects is in order. When mapping data sources to ontologies, one should take into account that sources store data, whereas instances of concepts are objects, where each object should be denoted by an ad hoc identifier (e.g., a constant in logic), not to be confused with any data item. In *DL-Lite*$_A$, we address this problem by keeping data value constants separate from object identifiers, and by accepting that object identifiers be created using data values, in particular as (logic) terms over data items. Note that this idea traces back to the work done in deductive object-oriented databases [Hul88].

To realize this idea, we modify the set Γ_O as follows. While Γ_V contains data value constants as before, Γ_O is built starting from Γ_V and a set Λ of function symbols of any arity (possibly 0), as follows: If $f \in \Lambda$, the arity of f is n, and $d_1, \ldots, d_n \in \Gamma_V$, then $f(d_1, \ldots, d_n)$ is a term in Γ_O, called *object term*. In other words, object terms are either objects constants (i.e., function symbols of arity 0), or function symbols applied to data value constants. In the following we call Γ_T the subset of Γ_O constituted by object

constants, i.e., object terms built with function symbols of arity 0. Also, we use Γ_{VT} to denote $\Gamma_V \cup \Gamma_T$.

To define the semantics of terms in Γ_O, we simply define an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as before, and we observe that the interpretation function $\cdot^{\mathcal{I}}$ now assigns a different element of $\Delta^{\mathcal{I}}_O$ to every object term (not only object constant) in Γ_O (i.e., we enforce the unique name assumption also on object terms).

Let us now turn our attention to the problem of linking data in the sources to objects in the ontology. To this end, we assume that data sources have been wrapped into a relational database *DB*. Note that this assumption is indeed realistic, as many data federation tools that provide exactly this kind of service are currently available. In this way, we can assume that all relevant data are virtually represented and managed by a relational data engine. In particular, one consequence is that we can query our data by using SQL. Formally, we assume that the database *DB* is characterized by a relational schema, and the corresponding extension. In particular, for each relation schema R, *DB* contains a set of tuples whose values are taken from Γ_{VT}. Note that, by virtue of this assumption, *DB* may store both data value constants and object constants. The evaluation of an SQL query φ over a database *DB*, denoted $ans(\varphi, DB)$, returns the set of tuples (of the arity of φ) of elements of Γ_{VT} that satisfy φ in *DB*.

To realize the link, we adapt principles and techniques from the literature on data integration [Len02]. In particular, we use the notion of *mapping*, which we now introduce by means of an example.

Example 6.10 Consider a *DL-Lite*$_A$ TBox in which *Person* is a concept name, *CITY-OF-BIRTH* is a role name, *age* and *cityName* are concept attributes names, and a relational database contains the ternary relation symbols *S1* and *S2* and the unary relation symbol *S3*. We want to model the situation where every tuple $(n, s, a) \in S1$ corresponds to a person whose name is n, whose surname is s, and whose age is a, and we want to denote such a person with $p(n, s)$. Note that this implies that we know that there are no two persons in our application that have the same pair (n, s) stored in *S1*. Similarly, we want to model the fact that every tuple $(n, s, cb) \in S2$ corresponds to a person whose name is n, whose surname is s, and whose city of birth is cb. Finally, we know that source *S3* directly stores object constants denoting instances of person. The following is the set of mapping assertions modeling the above situation.

$$S1(n, s, a) \rightsquigarrow Person(p(n, s)), age(p(n, s), a)$$
$$S2(n, s, cb) \rightsquigarrow CITY\text{-}OF\text{-}BIRTH(p(n, s), ct(cb)), cityName(ct(cb), cb)$$
$$S3(q) \rightsquigarrow Person(q).$$

Above, n, s, a, cb and q are variable symbols, p and ct are function symbols, whereas $p(n, s)$ and $ct(n)$ are so-called variable object terms (see below). ∎

The example shows that, in specifying mapping assertions, we need variable object terms, i.e., object terms containing variables. Indeed, we extend object terms to *variable object terms* by allowing also variables to appear in place of value constants.

We can now provide the definition of mapping assertions. Through a mapping we associate a conjunctive query over atomic concepts, domains, roles, attributes, and role

attributes (generically referred to as *predicates* in the following) with a first-order (more precisely, SQL) query of the appropriate arity over the database. The intuition is that, by evaluating such a query, we retrieve the facts that constitute the ABox assertions for the predicates appearing in the conjunctive query.

Definition 6.11 A *mapping assertion* is an assertion of the form

$$\varphi \rightsquigarrow \psi$$

where ψ is a *DL-Lite*$_A$ conjunctive query without existential variables and without constants, but whose atoms may contain variable object terms, and φ is an SQL query, i.e., an open first-order formula, over the database DB. ∎

We now describe the semantics of mapping assertions. To this end, we introduce the notion of ground instance of a formula. Let γ be a formula with free variables $\vec{x}$, and let $\vec{s}$ be a tuple of elements in Γ_{VT} of the same arity as $\vec{x}$. A ground instance $\gamma[\vec{x}/\vec{s}]$ of γ is obtained from γ by substituting every occurrence of x_i with s_i. We say that an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* the mapping assertion $\varphi \rightsquigarrow \psi$ wrt DB, if for every ground instance $\varphi[\vec{x}/\vec{s}] \rightsquigarrow \psi[\vec{x}/\vec{s}]$ of $\varphi \rightsquigarrow \psi$, we have that $ans(\varphi[\vec{x}/\vec{s}], DB) = \mathsf{true}$ implies $\psi[\vec{x}/\vec{s}]^{\mathcal{I}} = \mathsf{true}$ (where, for a ground atom $p(\vec{t})$, with $\vec{t} = (t_1, \ldots, t_n)$ a tuple of object-terms, we have that $p(\vec{t})^{\mathcal{I}} = \mathsf{true}$ if $(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \in p^{\mathcal{I}}$).

Finally, we can summarize the semantics of a *DL-Lite*$_A$ ontology with mapping assertions. Let DB be a database as defined above, $\mathcal{T}$ a *DL-Lite*$_A$ TBox, and $\mathcal{M}$ a set of mapping assertions between DB and $\mathcal{T}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a *model* of $\langle \mathcal{T}, \mathcal{M}, DB \rangle$ if $\mathcal{I}$ is a model of $\mathcal{T}$ and satisfies all mapping assertions in $\mathcal{M}$ wrt DB. The notion of certain answer to queries posed to $\langle \mathcal{T}, \mathcal{M}, DB \rangle$ remains the same as the one described in Section 6.1.

As we have seen in the previous section, both query answering and satisfiability in *DL-Lite*$_A$ reduce to query answering over an ABox seen as a database. In fact, the mappings define an ABox, which is the one obtained by instantiating the atoms in the conjunctive query on the right-hand side of mapping assertions with the constants retrieved by the SQL query on the left-hand side. We call such an ABox the ABox *induced* by the database DB and the mapping assertions $\mathcal{M}$ and denote it $ind(DB, \mathcal{M})$. Note that such constants may also instantiate variables appearing in variable object terms, giving rise to object terms in the atoms, and hence in the corresponding ABox. The object terms would not make any difference for satisfiability and query answering, and hence the above techniques still apply. However, we can avoid materializing such an ABox by proceeding as follows. First, we split each mapping assertion $\varphi \rightsquigarrow \psi$ into several assertions of the form $\varphi \rightsquigarrow p$, one for each atom p in ψ. Then, we unify in all possible ways the atoms in the query q to be evaluated with the right-hand side atoms of the (split) mappings, thus obtaining a (bigger) union of conjunctive queries containing variable object terms. Then, we unfold each atom with the corresponding left-hand side mapping query. Observe that, after unfolding, all variable object terms disappear, except those that are returned by q as instantiations of free variables. Hence, what we have obtained is a FOL query to the database whose answer then has to be processed in order to generate the object terms to be returned. Notice that no post-processing is necessary in the case where the query q does not contain any free object variable.

Example. Refer to the previous example, and consider now the following query over the TBox, asking for the age of those people that are born in Rome:

$$q(z) \leftarrow \exists x, y.\ \mathit{Person}(x), \mathit{CITY\text{-}OF\text{-}BIRTH}(x, y), \mathit{cityName}(y, \mathsf{Roma}), \mathit{age}(x, z).$$

Let us, for simplicity, assume that no reasoning on the TBox has to be done in order to answer the query q, and hence let us directly evaluate such a query by exploiting the mapping, without materializing the ABox of the KB.

We first split the mapping (left as an exercise), and then unify the atoms in the query with the right-hand side atoms in the split mapping, thus obtaining

$$\begin{aligned} q(z) \leftarrow\ & \mathit{Person}(p(n, s)), \mathit{CITY\text{-}OF\text{-}BIRTH}(p(n, s), \mathit{ct}(\mathsf{Roma})), \\ & \mathit{cityName}(\mathit{ct}(\mathsf{Roma}), \mathsf{Roma}), \mathit{age}(p(n, s), z). \end{aligned}$$

Then, we unfold each atom with the corresponding left-hand side mapping query, thus obtaining

$$q(z) \leftarrow \mathit{S2}(n, s, \mathsf{Roma}), \mathit{S1}(n, s, z),$$

where w is a new (existential) variable symbol. The obtained query can be then simply evaluated over the database in order to get the certain answers to q.

6.5 *DL-Lite*$_A$ as an instantiation of the TONES framework for situated ontologies

Given the representation of a *DL-Lite*$_A$ ABox as a database DB described in the previous section, we now define a formalism $\langle \mathcal{F}, \Sigma_{\mathcal{E}}, \mathcal{L}_{\mathcal{E}}, \mathit{Sem}_{\mathcal{E}} \rangle$ for situated ontologies (for DB) that corresponds to *DL-Lite*$_A$. Its components are defined as follows.

- $\mathcal{F}$ is the ontology formalism $\mathcal{F}_{\mathit{DL\text{-}Lite}_{FR}}$ defined in Section 6.2.
- $\Sigma_{\mathcal{E}}$ is the alphabet of DB relation symbols (those that may occur in the left-hand side of mapping assertions).
- $\mathcal{L}_{\mathcal{E}}$ is a language for the mapping between ontologies and DB, where each element of $\mathcal{L}_{\mathcal{E}}$ consists of a set of mapping assertions of the form introduced in Definition 6.11.
- $\mathit{Sem}_{\mathcal{E}}$ is a pair $(\mathcal{S}_{\mathcal{E}}, \delta_{\mathcal{E}})$, where
  - $\mathcal{S}_{\mathcal{E}}$ is the set of all database instances constructible over the relations in $\Sigma_{\mathcal{E}}$.
  - $\delta_{\mathcal{E}}$ is a function that, given a database DB, a *DL-Lite*$_A$ TBox $\mathcal{T}$, a set $\mathcal{M}$ of mapping assertions between DB and $\mathcal{T}$, and an interpretation $\mathcal{I}$, returns true if $\mathcal{I}$ is a model of $\mathcal{T}$ and satisfies all mapping assertions in $\mathcal{M}$ wrt DB, false otherwise.

In terms of the framework for situated ontologies, the *query answering* service has, at the syntactic level:

- A signature specifying a name for the service, e.g., *certain*, and the fact that there are 4 arguments as follows:

  - A *DL-Lite*$_A$ TBox $\mathcal{T}$.
  - A database DB.
  - A set $\mathcal{M}$ of mapping assertions between DB and $\mathcal{T}$.
  - A query (CQ or UCQ) $q(\vec{x})$ over $\mathcal{T}$.
- A precondition imposing that the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be satisfiable, where $\mathcal{A} = ind(DB, \mathcal{M})$ is the ABox induced by the database DB and the mapping assertions $\mathcal{M}$.
- The domain for the result, defined as $\{\mathsf{true}, \mathsf{false}\} \cup 2^{\mathcal{D}}$, where $\mathcal{D}$ is the set of all tuples of elements of $\Gamma_V \cup \Gamma_O$.

The meaning of this service is to return the *certain answers* to $q(\vec{x})$ over $\mathcal{T}$ and DB wrt. $\mathcal{M}$, i.e., all tuples $\vec{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that, when substituted to the variables $\vec{x}$ in $q(\vec{x})$, we have that $\langle \mathcal{T}, ind(DB, \mathcal{M}) \rangle \models q(\vec{t})$.

7 Ontology based peer-to-peer data integration systems

In this section, we provide possible instantiations for the TONES framework considering the setting of peer ontologies. We recall that a peer ontology formalism is a triple $\mathcal{F}_P = \langle \{\mathcal{F}^i\}_{1 \le i \le k}, \{\mathcal{L}_M^{ij}\}_{1 \le i,j \le k}, Sem_P \rangle$, where each $\mathcal{F}^i$ is an ontology formalism, each $\mathcal{L}_M^{ij}$ is a mapping language, and Sem_P is a semantic specification for peer ontologies. Our aim here is to describe how such a formalism $\mathcal{F}_P$ can be instantiated, i.e., we give concrete examples for each $\mathcal{F}^i$, each $\mathcal{L}_M^{ij}$, and Sem_P.

Before showing the instantiation of the TONES framework, we describe a framework for peer-to-peer (P2P) data integration that was recently presented in [CDGL$^+$06b], suitably adapted to the case in which peers in the system export knowledge specified in terms of ontologies. Specifically, we discuss two alternative approaches to interpretation of P2P data integration systems, and compare them to each other, taking into account some desirable properties in the context of P2P data management.

7.1 Ontology based peer-to-peer data integration systems

In a *P2P data integration system* [HIST03, BGK$^+$02, FKLS03, CDGLR04], each peer is essentially a mediator-based data integration system, i.e., it manages a set of local data sources (whose schema is called *local source schema*) semantically connected, via a *local mapping*, to a (virtual) global schema called the *peer schema*, which can be expressed in terms of an ontology language, e.g., a DL TBox. In other words each peer in the system is actually a situated ontology, whose environment is constitutes by the local source schema. In addition, the specification of a peer includes a set of *P2P mappings* (also called peer mappings) that specify the semantic relationships with the data exported by other peers, as shown in Figure 1. Information in such systems can be *queried* to any peer (by external users or other peers). The queried peer, by exploiting its P2P mappings, can make use of the data in the other peers for providing the answer.

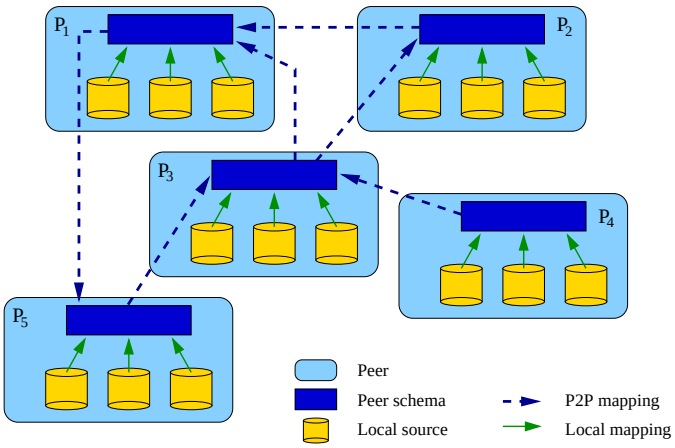

Figure 1: Peer-to-Peer Data Integration System

7.1.1 Formal framework for ontology-based P2P data integration

We refer to a fixed, infinite, denumerable set Γ of constants. Such constants are shared by all peers, and denote the data items managed by the ontoloty-based Peer-to-Peer Data Integration System (P2PDIS). Moreover, given an alphabet A, we denote with $\mathcal{L}_A$ the set of function-free first-order logic (FOL) formulas whose relation symbols are in A and whose constants are in Γ.

We also consider conjunctive queries, i.e., SQL select-project-join queries. Formally, a *conjunctive query* (CQ) of arity n over A is a query written in the form

$$\{\mathbf{x} \mid \exists \mathbf{y}.\mathit{body}_{cq}(\mathbf{x}, \mathbf{y})\}$$

where $\mathit{body}_{cq}(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of $\mathcal{L}_A$ involving the free variables (also called the *distinguished* variables of the query) $\mathbf{x} = x_1, \ldots, x_n$, the existentially quantified variables (also called the *non-distinguished* variables of the query) $\mathbf{y} = y_1, \ldots, y_m$, and constants from Γ.

An *ontology-based P2P data integration system* $\mathcal{P} = \{P_1, \ldots, P_n\}$ is constituted by a set of n peers. Each peer $P_i \in \mathcal{P}$ (cf. [HIST03]) is defined as a tuple $P_i = \langle O, S, L, M, \mathcal{L} \rangle$, where:

- O is the *(global) ontology* of P_i, which is a finite set of formulas of $\mathcal{L}_{A_O}$ (representing the ontology language), where A_O is an alphabet (disjoint from the other alphabets in $\mathcal{P}$) called the *alphabet* of P_i. Notice that O is expressed in some fragment of first-order logic. Notable cases are those in which O is expressed in terms of DLs. In these cases, since the aim of O is in general to provide an intensional view of the information managed by the peer, global ontologies are considered that are constituted by the TBox only.

- S is the *(local) source schema* of P_i, which is simply a finite relational alphabet (again disjoint from the other alphabets in $\mathcal{P}$), called the *local alphabet* of P_i. Intuitively, the source schema describes the structure of the data sources of the peer (possibly obtained by wrapping physical sources), i.e., the sources where the real data managed by the peer are stored.

- L is a set of *(local) mapping assertions* between O and S. Each local mapping assertion is an expression of the form

$$cq_S \rightsquigarrow cq_O,$$

where cq_S and cq_O are two conjunctive queries of the same arity, respectively over the source schema S and over the peer ontology O. The local mapping assertions establish the connection between the elements of the source schema and those of the peer ontology in P_i. In particular, an assertion of the form $cq_S \rightsquigarrow cq_O$ specifies that all the data satisfying the query cq_S over the sources also satisfy the concept in the peer ontology represented by the query cq_O. In the terminology used in data integration, the combination of peer ontology, source schema, and local mapping assertions constitutes a GLAV *data integration system* [Len02] managing a set of sound data sources defined in terms of a (virtual) global schema, whereas in the terminology used in the TONES framework, the above combination represents a situated ontology.

- M is a set of *peer mapping assertions*, which specify the semantic relationships that the peer P_i has with the other peers. Each assertion in M is an expression of the form

$$cq' \rightsquigarrow cq,$$

where cq, called the *head* of the assertion, is a conjunctive query over the (ontology of the) peer P_i, while cq', called the *tail* of the assertion, is a conjunctive query of the same arity as cq over (the ontology of) one of the other peers in $\mathcal{P}$. A peer mapping assertion $cq' \rightsquigarrow cq$ from peer P_j to peer P_i expresses the fact that the P_j-concept represented by cq' is mapped to the P_i-concept represented by cq. From an extensional point of view, the assertion specifies that every tuple that can be retrieved from P_j by issuing query cq' satisfies cq in P_i. Observe that no limitation is imposed on the topology of the whole set of peer mapping assertions in the system $\mathcal{P}$, and hence the set of all peer mappings may be cyclic.

- $\mathcal{L}$ is a relational query language specifying the class of queries that the peer P_i can process. We assume that $\mathcal{L}$ is some fragment of FOL that accepts at least conjunctive queries. We say that the queries in $\mathcal{L}$ are those *accepted by* P_i. Notice that this implies that, for each peer mapping assertion $cq' \rightsquigarrow cq$ from another peer P_j to peer P_i in M, we have that cq' is accepted by P_j.

An *extension* for a P2PDIS $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set $\mathcal{D} = \{D_1, \ldots, D_n\}$, where each D_i is an extension of the predicates in the local source schema of peer P_i.

A P2PDIS, together with an extension, is intended to be queried by external users. A user enquires the whole system by accessing any peer P of $\mathcal{P}$, and by issuing a *query* q to P. The query q is processed by P if and only if q is expressed over the ontology of P and is accepted by P.

Example 7.1 Let us consider the P2PDIS in Figure 2, in which we have 4 peers P_1, P_2, P_3, and P_4. To maintain things simple, ontologies of in this example are relational schema with only key constraints specified over relation symbols [AHV95].

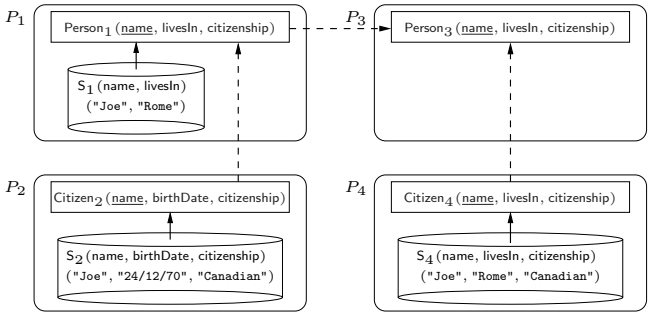

Figure 2: The P2P Data Integration System of Example 7.1

The ontology of peer P_1 is formed by a relation schema $\mathsf{Person}_1(\underline{\mathsf{name}}, \mathsf{livesIn}, \mathsf{citizenship})$, where name is the key (we underline the key of a relation). P_1 contains a local source $\mathsf{S}_1(\mathsf{name}, \mathsf{livesIn})$, mapped to the global ontology by the assertion $\{x, y \mid \mathsf{S}_1(x, y)\} \rightsquigarrow \{x, y \mid \exists z.\mathsf{Person}_1(x, y, z)\}$. Moreover, it has a peer mapping assertion $\{x, z \mid \exists y.\mathsf{Citizen}_2(x, y, z)\} \rightsquigarrow \{x, z \mid \exists y.\mathsf{Person}_1(x, y, z)\}$ relating information in peer P_2 to those in peer P_1.

P_2 has $\mathsf{Citizen}_2(\underline{\mathsf{name}}, \mathsf{birthDate}, \mathsf{citizenship})$ as global ontology, and a local source $\mathsf{S}_2(\mathsf{name}, \mathsf{birthDate}, \mathsf{citizenship})$ mapped to the global ontology through the local mapping $\{x, y, z \mid \mathsf{S}_2(x, y, z)\} \rightsquigarrow \{x, y, z \mid \mathsf{Citizen}_2(x, y, z)\}$. P_2 has no peer mappings.

P_3 has $\mathsf{Person}_3(\underline{\mathsf{name}}, \mathsf{livesIn}, \mathsf{citizenship})$ as global ontology, contains no local sources, and has a peer mapping $\{x, y, z \mid \mathsf{Person}_1(x, y, z)\} \rightsquigarrow \{x, y, z \mid \mathsf{Person}_3(x, y, z)\}$ with P_1, and a peer mapping $\{x, y, z \mid \mathsf{Citizen}_4(x, y, z)\} \rightsquigarrow \{x, y, z \mid \mathsf{Person}_3(x, y, z)\}$ with P_4.

P_4 has $\mathsf{Citizen}_4(\underline{\mathsf{name}}, \mathsf{livesIn}, \mathsf{citizenship})$ as global ontology, and a local source $\mathsf{S}_4(\mathsf{name}, \mathsf{livesIn}, \mathsf{citizenship})$ mapped to the global ontology through the local mapping $\{x, y, z \mid \mathsf{S}_4(x, y, z)\} \rightsquigarrow \{x, y, z \mid \mathsf{Citizen}_4(x, y, z)\}$. P_4 has no peer mappings.

Finally, Figure 2 shows also an extension of the system, which includes $\mathsf{S}_1(\texttt{"Joe"}, \texttt{"Rome"})$, $\mathsf{S}_2(\texttt{"Joe"}, \texttt{"24/12/70"}, \texttt{"Canadian"})$, and $\mathsf{S}_4(\texttt{"Joe"}, \texttt{"Rome"}, \texttt{"Canadian"})$. ∎

7.1.2 Classical semantics for ontology-based P2P data integration systems

In this section we present a logical formalization of ontology-based P2P data integration systems based on classical first-order logic. Such a formalization is the first one that has been proposed for P2P data integration [CL93, Koc02, HIST03].

We assume that the peers are interpreted over a fixed infinite domain Δ. We also fix the interpretation of the constants in Γ (cf. previous section) so that: (*i*) each $c \in \Gamma$ denotes an element $d \in \Delta$; (*ii*) different constants in Γ denote different elements of Δ; (*iii*) each element in Δ is denoted by a constant in Γ.[6] It follows that Γ is actually isomorphic to Δ, so that we can use (with some abuse of notation) constants in Γ whenever we want to denote domain elements.

[6] In other words the constants in Γ act as *standard names* [LL01].

Semantics of one peer We focus first on the semantics of a single peer $P = \langle O, S, L, M, \mathcal{L} \rangle$. Let us call *peer theory of* P the FOL theory T_P defined as follows. The alphabet of T_P is obtained as union of the alphabet A_O of O and the alphabet of the local sources S of P. The axioms of T_P are the formulas in O plus one formula of the form

$$\forall \mathbf{x}.(\exists \mathbf{y}.body_{cq_S}(\mathbf{x}, \mathbf{y}) \supset \exists \mathbf{z}.body_{cq_O}(\mathbf{x}, \mathbf{z}))$$

for each local mapping assertion $cq_S \rightsquigarrow cq_O$ in L.

Observe that the peer mapping assertions of P are not considered in T_P, and that T_P is an "open theory", since for the sources in P we only have the schema, S, and not the extension. We call *local source database* for P, a database D for the source schema S, i.e., a finite relational interpretation of the relation symbols in S. An interpretation $\mathcal{I}$ of T_P is a *model of* P *based on* D if it is a model of the FOL theory T_P such that for each relational symbol $s \in S$, we have that $s^{\mathcal{I}} = s^D$.

Finally, consider a query q of arity n, expressed in the query language $\mathcal{L}$ accepted by P. Given an interpretation $\mathcal{I}$ of T_P, we denote with $q^{\mathcal{I}}$ the set of n-tuples of constants in Γ obtained by evaluating q in $\mathcal{I}$, according to the semantics of $\mathcal{L}$. We define the *certain answers* $ANS(q, P, D)$ to q (accepted by P) based on a local source database D for P, as the set of tuples $\mathbf{t}$ of constants in Γ such that for all models $\mathcal{I}$ of P based on D, we have that $\mathbf{t} \in q^{\mathcal{I}}$.

Semantics for ontology-based P2P data integration systems Based on the above logical formalization of a peer, we now present the "classical" approach to providing a semantics to the whole ontology-based P2P data integration system. The classical approach is what we may call the FOL approach, followed by [CL93, Koc02, HIST03]. In this approach, one associates to a P2PDIS $\mathcal{P}$ a *single* (open) FOL theory $T_{\mathcal{P}}$, obtained as the disjoint union of the various peer theories (peer mappings are not considered in $T_{\mathcal{P}}$).

By following the approach used for a single peer, we consider a *source database* $\mathcal{D}$ *for* $\mathcal{P}$, simply as the (disjoint) union of one local source database D for each peer P in $\mathcal{P}$. We call *FOL interpretation of* $T_{\mathcal{P}}$ *based on* $\mathcal{D}$ an interpretation $\mathcal{I}$ of the FOL theory $T_{\mathcal{P}}$ such that for each relational symbol s of the source schemas in the peers of $\mathcal{P}$, we have that $s^{\mathcal{I}} = s^{\mathcal{D}}$. Furthermore, we assume that all interpretations have the same (fixed infinite) interpretation domain Δ, and that different constants in Σ_o are interpreted with the same different objects of Δ, in each interpretation (i.e., we assume constants in Σ_o acting as standard names).

Then we call *FOL model of* $\mathcal{P}$ *based on* $\mathcal{D}$ a model $\mathcal{I}$ of $T_{\mathcal{P}}$ based on $\mathcal{D}$ that is also a model of the formula

$$\forall \mathbf{x}.(\exists \mathbf{y}.body_{cq_1}(\mathbf{x}, \mathbf{y}) \supset \exists \mathbf{z}.body_{cq_2}(\mathbf{x}, \mathbf{z}))$$

for each peer mapping assertion $cq_1 \rightsquigarrow cq_2$ in the peers of $\mathcal{P}$.

Finally, given a query q over the peer P_i in $\mathcal{P}$ and a source database $\mathcal{D}$ for $\mathcal{P}$, we define the *certain answers* $ANS_{fol}(q, P_i, \mathcal{P}, \mathcal{D})$ to q in $\mathcal{P}$ based on $\mathcal{D}$ under FOL semantics, as the set of tuples $\mathbf{t}$ of constants in Γ such that for every FOL model $\mathcal{I}$ of $\mathcal{P}$ based on $\mathcal{D}$, we have that $\mathbf{t} \in q^{\mathcal{I}}$.

7.1.3 Limitations of first-order approaches

Although correct from a formal point of view, the usual approach of resorting to a first-order logic interpretation of peer mappings, which we have described in the above section, has several drawbacks, both from the modeling and from the computational perspective. Consider, for example, three central desirable properties of P2P systems:

- *Modularity*: i.e., how autonomous are the various peers in a P2P system with respect to the semantics. Indeed, since each peer is autonomously built and managed, it should be clearly interpretable both alone and when involved in interconnections with other peers. In particular, interconnections with other peers should not radically change the interpretation of the concepts expressed in the peer.

- *Generality*: i.e., how free we are in placing connections (peer mappings) between peers. This is a fundamental property, since actual interconnections among peers are not under the control of any actor in the system.

- *Decidability*: i.e., are sound, complete and terminating query answering mechanisms available? If not, it becomes critical to establish basic quality assurance of the answers returned by the system.

Actually, these desirable properties are weakly supported by approaches based directly on FOL semantics. Indeed, such approaches essentially consider the P2P system as a single flat logical theory. As a result, the structure of the system in terms of peers is lost and remote interconnections may propagate constraints that have a deep impact on the semantics of a peer. Moreover, under arbitrary P2P interconnections, query answering under the first-order semantics is undecidable, even when the single peers have an extremely restricted structure. Motivated by these observations, several authors proposed suitable limitations to the form of P2P mappings, such as acyclicity, thus giving up generality to retain decidability [HIST03, Koc02, FKMP03].

To overcome the above drawbacks, we recall here the proposal of a new semantics for P2P systems, with the following aims:

- We want to take into account that peers in our context are to be considered autonomous sites that exchange information. In other words, peers are modules, and the modular structure of the system should be explicitly reflected in the definition of its semantics.

- We do not want to limit a-priori the topology of the mapping assertions among the peers in the system. In particular, we do not want to impose acyclicity of assertions.

- We seek for a semantic characterization that leads to a setting where query answering is decidable, and possibly, polynomially tractable.

We base our proposal of a new semantics for P2P systems on epistemic logic, and we show that the new semantics is clearly superior to the usual FOL semantics with respect to all three properties mentioned above.

7.1.4 Multi-modal epistemic formalization

In this section we present a logical formalization of ontology-based P2P data integration systems based on the use of epistemic logic. Such a formalization brings several advantages w.r.t. classical FOL formalizations, as discussed in the above subsection. In particular, we adopt a *multi-modal* epistemic logic, based on the premise that each peer in the system can be seen as a rational agent. More precisely, the formalization we provide in this section is based on **K**, the multi-modal version of the modal logic S5 [Che80, LL01].

The logic K. The language $\mathcal{L}(\mathbf{K})$ of **K** is obtained from first-order logic by adding a set $\mathbf{K_1}, \ldots, \mathbf{K_n}$ of modal operators, for the forming rule: if ϕ is a (possibly open) formula, then also $\mathbf{K_i}\phi$ is so, for $1 \leq i \leq n$ for a fixed n. In **K**, each modal operator is used to formalize the epistemic state of a different agent. Informally, the formula $\mathbf{K_i}\phi$ should be read as "ϕ is known to hold by the agent i". The semantics of **K** is such that what is known by an agent must hold in the real world: in other words, the agent cannot have inaccurate knowledge of what is true, i.e., believe something to be true although in reality it is false. Moreover, **K** states that the agent has complete information on what it knows, i.e., if agent i knows ϕ then it knows of knowing ϕ, and if agent i does not know ϕ, then it knows that it does not know ϕ. In other words, the following assertions hold for every **K** formula ϕ:

$\mathbf{K_i}\phi \supset \phi$	known as the axiom schema T
$\mathbf{K_i}\phi \supset \mathbf{K_i}(\mathbf{K_i}\phi)$	known as the axiom schema 4
$\neg\mathbf{K_i}\phi \supset \mathbf{K_i}(\neg\mathbf{K_i}\phi)$	known as the axiom schema 5

To define the semantics of **K**, we start from first-order interpretations. As done in Section 7.1.2, we restrict our attention to first-order interpretations that share a fixed infinite domain Δ and assume that constants of the set Γ act as standard names for Δ.

Formulas of **K** are interpreted over **K**-structures. A **K**-*structure* is a Kripke structure E of the form $(W, \{R_1, \ldots R_n\}, V)$, where: W is a set whose elements are called *possible worlds*; V is a function assigning to each $w \in W$ a first-order interpretation $V(w)$; and each R_i, called the *accessibility relation* for the modality $\mathbf{K_i}$, is a binary relation over W, with the following constraints:

if $w \in W$ then $(w, w) \in R_i$, i.e., R_i is reflexive
if $(w_1, w_2) \in R_i$ and $(w_2, w_3) \in R_i$ then $(w_1, w_3) \in R_i$, i.e., R_i is transitive
if $(w_1, w_2) \in R_i$ and $(w_1, w_3) \in R_i$ then $(w_2, w_3) \in R_i$, i.e., R_i is euclidean

An **K**-*interpretation* is a pair E, w, where $E = (W, \{R_1, \ldots R_n\}, V)$ is an **K**-structure, and w is a world in W. We inductively define when a sentence (i.e., a closed formula) ϕ *is true in an interpretation* E, w (or, is true on world $w \in W$ in E), written $E, w \models \phi$, as

follows:[7]

$$
\begin{array}{lll}
E, w \models P(c_1, \ldots, c_n) & \text{iff} & V(w) \models P(c_1, \ldots, c_n) \\
E, w \models \phi_1 \wedge \phi_2 & \text{iff} & E, w \models \phi_1 \text{ and } E, w \models \phi_2 \\
E, w \models \neg\phi & \text{iff} & E, w \not\models \phi \\
E, w \models \exists x.\psi & \text{iff} & E, w \models \psi_c^x \text{ for some constant } c \\
E, w \models \mathbf{K_i}\phi & \text{iff} & E, w' \models \phi \text{ for every } w' \text{ such that } (w, w') \in R_i
\end{array}
$$

We say that a sentence ϕ is *satisfiable* if there exists an **K**-*model* for ϕ, i.e., an **K**-interpretation E, w such that $E, w \models \phi$, *unsatisfiable* otherwise. A *model* for a set Σ of sentences is a model for every sentence in Σ. A sentence ϕ is *logically implied* by a set Σ of sentences, written $\Sigma \models_{\mathbf{K}} \phi$, if and only if in every **K**-model E, w of Σ, we have that $E, w \models \phi$.

Notice that, since each accessibility relation of a **K**-structure is reflexive, transitive and Euclidean, all instances of axiom schemas T, 4 and 5 are satisfied in every **K**-interpretation.

Epistemic semantics for ontology-based P2P data integration systems. Due to the characteristics mentioned above, **K** is well-suited to formalize P2PDISs of the kind presented in Section 2. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a P2PDIS in which each peer P_i has identifier i. For each peer $P_i = \langle O, S, L, M, \mathcal{L}\rangle$ we define the theory $\mathcal{T}_K(P_i)$ in **K** as the union of the following sentences:

- Global ontology O of P_i: for each sentence ϕ in O, we have

  $$\mathbf{K_i}\phi$$

  Observe that ϕ is a first-order sentence expressed in the alphabet of P_i, which is disjoint from the alphabets of all the other peers in $\mathcal{P}$.

- Local mapping assertions L between O and the local source schema S: for each mapping assertion $\{\mathbf{x} \mid \exists \mathbf{y}.body_{cq_S}(\mathbf{x}, \mathbf{y})\} \rightsquigarrow \{\mathbf{x} \mid \exists \mathbf{z}.body_{cq_O}(\mathbf{x}, \mathbf{z})\}$ in L, we have

  $$\mathbf{K_i}(\forall \mathbf{x}.\exists \mathbf{y}.body_{cq_S}(\mathbf{x}, \mathbf{y}) \supset \exists \mathbf{z}.body_{cq_O}(\mathbf{x}, \mathbf{z}))$$

- peer mapping assertions M: for each peer mapping assertion $\{\mathbf{x} \mid \exists \mathbf{y}.body_{cq_j}(\mathbf{x}, \mathbf{y})\} \rightsquigarrow \{\mathbf{x} \mid \exists \mathbf{z}.body_{cq_i}(\mathbf{x}, \mathbf{z})\}$ between the peer j and the peer i in M, we have

  $$\forall \mathbf{x}.\mathbf{K_j}(\exists \mathbf{y}.body_{cq_j}(\mathbf{x}, \mathbf{y})) \supset \mathbf{K_i}(\exists \mathbf{z}.body_{cq_i}(\mathbf{x}, \mathbf{z})) \quad (11)$$

  In words, this sentence specifies the following rule: for each tuple of values $\mathbf{t}$, if peer j knows the sentence $\exists \mathbf{y}.body_{cq_j}(\mathbf{t}, \mathbf{y})$, then peer i knows the sentence $\exists \mathbf{z}.body_{cq_i}(\mathbf{t}, \mathbf{z})$ holds.

We denote by $\mathcal{T}_K(\mathcal{P})$ the theory corresponding to the P2PDIS $\mathcal{P}$, i.e., $\mathcal{T}_K(\mathcal{P}) = \bigcup_{i=1,\ldots,n} \mathcal{T}_K(P_i)$.

[7] We have used ψ_c^x to denote the formula obtained from ψ by substituting each free occurrence of the variable x with the constant c.

Example 7.2 We provide now the formalization of the P2PDIS of Example 7.1. The theory $\mathcal{T}_K(P_1)$ modeling peer P_1 is the conjunction of:

$$\mathbf{K_1}(\forall x, y, y', z, z'.\mathsf{Person}_1(x, y, z) \wedge \mathsf{Person}_1(x, y', z') \supset y = y' \wedge z = z')$$
$$\mathbf{K_1}(\forall x, y.\mathsf{S}_1(x, y) \supset \exists z.\mathsf{Person}_1(x, y, z))$$
$$\forall x, z.\mathbf{K_2}(\exists y.\mathsf{Citizen}_2(x, y, z)) \supset \mathbf{K_1}(\exists y.\mathsf{Person}_1(x, y, z))$$

The theory $\mathcal{T}_K(P_2)$ modeling peer P_2 is the conjunction of:

$$\mathbf{K_2}(\forall x, y, y', z, z'.\mathsf{Citizen}_2(x, y, z) \wedge \mathsf{Citizen}_2(x, y', z') \supset y = y' \wedge z = z')$$
$$\mathbf{K_2}(\forall x, y, z.\mathsf{S}_2(x, y, z) \supset \mathsf{Citizen}_2(x, y, z))$$

The theory $\mathcal{T}_K(P_3)$ modeling peer P_3 is the conjunction of:

$$\mathbf{K_3}(\forall x, y, y', z, z'.\mathsf{Person}_3(x, y, z) \wedge \mathsf{Person}_3(x, y', z') \supset y = y' \wedge z = z')$$
$$\forall x, y.\mathbf{K_1}(\exists z.\mathsf{Person}_1(x, z, y)) \supset \mathbf{K_3}\exists z.\mathsf{Person}_3(x, z, y)$$
$$\forall x, y, z.\mathbf{K_4}(\mathsf{Citizen}_4(x, y, z)) \supset \mathbf{K_3}\mathsf{Person}_3(x, y, z)$$

The theory $\mathcal{T}_K(P_4)$ modeling peer P_4 is the conjunction of:

$$\mathbf{K_4}(\forall x, y, y', z, z'.\mathsf{Citizen}_4(x, y, z) \wedge \mathsf{Citizen}_4(x, y', z') \supset y = y' \wedge z = z')$$
$$\mathbf{K_4}(\forall x, y, z.\mathsf{S}_4(x, y, z) \supset \mathsf{Citizen}_4(x, y, z))$$

∎

The extension $\mathcal{D} = \{D_1, \ldots, D_n\}$ of a P2PDIS $\mathcal{P}$ is modeled as a sentence constituted by the conjunction of all facts corresponding to the tuples stored in the sources, i.e., $DB(\mathcal{D}) = \bigwedge_{i=1}^{n} DB(D_i)$ where $DB(D_i) = \mathbf{K_i}(\bigwedge_{t \in r^{D_i}} r(t))$.

A client of the P2PDIS interacts with one of the peers, say peer P_i, posing a *query* to it. A query q is an open formula $q(\mathbf{x})$ with free variables $\mathbf{x}$ expressed in the language accepted by the peer P_i (we recall that such a language is a subset of first-order logic). The semantics of a query $q \in \mathcal{L}$ posed to a peer $P_i = \langle O, S, L, M, \mathcal{L} \rangle$ of $\mathcal{P}$ with respect to an extension $\mathcal{D}$ is defined as the set of tuples

$$ANS_{\mathbf{K}}(q, P_i, \mathcal{P}, \mathcal{D}) = \{\mathbf{t} \mid \mathcal{T}_K(\mathcal{P}) \cup DB(\mathcal{D}) \models_{\mathbf{K}} \mathbf{K_i} q(\mathbf{t})\}$$

where $q(\mathbf{t})$ denotes the sentence obtained from the open formula $q(\mathbf{x})$ by replacing all occurrences of the free variables in $\mathbf{x}$ with the corresponding constants in $\mathbf{t}$.

7.2 Ontology-based data integration systems as instantiations of the TONES framework for peer ontologies

We now show how the TONES framework for peer ontologies presented in Section 2.3 can be instantiated to capture the P2P system formalizations presented above. In presenting such an instantiation, we also assume that global ontologies are specified in a DL belonging to the *DL-Lite*-family, which is particularly suited for effective ontology-based data access (see also Section 6).

7.2.1 Instantiation of the framework for situated ontologies

First, we need to consider each ontology formalism in $\mathcal{F}_P$ to be a situated ontology formalism, i.e., $\mathcal{F}^i = \langle F^i, \mathcal{L}^i_{\mathcal{E}}, Sem^i_{\mathcal{E}} \rangle$, where (i) F^i is an ontology formalism, (ii) $\mathcal{L}^i_{\mathcal{E}}$ is a language over $\Sigma^i \cup \Sigma^i_{\mathcal{E}}$ for the mapping between $\mathcal{F}^i$ and the environment $\mathcal{E}^i$ in which it is situated, where Σ^i is the alphabet of F^i and $\Sigma^i_{\mathcal{E}}$ is the alphabet of the environment, and (iii) $Sem^i_{\mathcal{E}} = \langle \mathcal{S}^i_{\mathcal{E}}, \delta^i_{\mathcal{E}} \rangle$ is the semantic specification for the mapping, where $\mathcal{S}^i_{\mathcal{E}}$ is a set of states for the environment, and $\delta^i_{\mathcal{E}}$ is a function associating a truth value to an F^i-ontology $\mathcal{O}^i$, given an interpretation for $\mathcal{O}^i$, and a state $E^i \in \mathcal{S}^i_{\mathcal{E}}$.

We assume also that each situated ontology formalism $\mathcal{F}^i$ is instantiated in a such a way that:

1. The $\mathcal{F}^i$-ontology $\mathcal{O}^i$ is a *DL-Lite* ontology without the ABox, i.e., $\mathcal{O}^i$ is actually a TBox specified in one of the DLs belonging to the *DL-Lite* family [CDGL$^+$06a] (we refer the reader to Section 6 for details on such an instantiation). We also assume that all ontology alphabets Σ^i share the same countable subset Σ_o of object-constants (we do not consider value-constant here), i.e., $\Sigma^i_o = \Sigma_o$ for $1 \leq i \leq k$ (notice that Σ_o corresponds to the set Γ defined in Section 7.1.2).

2. The environment $\mathcal{E}^i$ is a relational database schema with relational alphabet $\Sigma^i_{\mathcal{E}}$, (possibly) representing a set of (wrapped) data sources, and the language $\mathcal{L}^i_{\mathcal{E}}$ for the mapping allows for constructing a mapping $\mathcal{M}^i$ constituted by a set of assertions, each of the form
$$\varphi \rightsquigarrow \psi$$
where ψ is a conjunctive query specified over the ontology $\mathcal{O}^i$, and φ is a conjunctive query of the same arity as that of ψ, specified over the relational database schema constituting the environment (notice that φ and ψ play the same role of cq_S and cq_O in local mapping assertions of a peer in a P2PDIS described in Section 7.1.2).

3. The semantic specification for the mapping $Sem^i_{\mathcal{E}} = \langle \mathcal{S}^i_{\mathcal{E}}, \delta^i_{\mathcal{E}} \rangle$ is such that, the set $\mathcal{S}^i_{\mathcal{E}}$ of states of the environment is a set of database instances for the database schema $\mathcal{E}^i$, and, given a state $E^i \in \mathcal{S}^i_{\mathcal{E}}$ and an interpretation $\mathcal{I}^i$ for $\mathcal{O}^i$,
$$\delta^i_{\mathcal{E}}(\mathcal{O}^i, \mathcal{I}^i, E^i) = \mathsf{true} \text{ iff } \varphi^{\mathcal{E}^i} \subseteq \psi^{\mathcal{I}^i} \text{ for each } \varphi \rightsquigarrow \psi \in \mathcal{M}^i$$
where $\varphi^{\mathcal{E}^i}$ denotes the evaluation of the query φ over the database instance $\mathcal{E}^i$, and $\psi^{\mathcal{I}^i}$ denotes the evaluation of the query ψ over the interpretation $\mathcal{I}^i$.

In other words, in terms of the terminology used in data integration [Len02], each instantiation of each ontology formalism, i.e., each peer, in the system for $\mathcal{F}_P$-peer ontologies is actually a data integration system exporting a *DL-Lite* TBox as global reconciled schema, adopting a GLAV approach to specify the mapping, according to which queries over the global ontology are associated with queries over the sources, and using the language of conjunctive queries to specify queries in the mapping.

We point out that also more complex instantiations of a situated ontology formalism may be adopted for a peer, in particular those that face with the problem of the "mismatch" existing between the objects that are instances of the global ontology and the

tuples of values that are instances of the relational database at the sources. For further details on this matter, we refer the reader to Section 6, where an ontology formalism instantiation is described, which allows for not storing object identifiers in the underlying database (which instead is required by the present instantiation).

7.2.2 Instantiation of mapping languages

We consider now mapping languages and provide a possible instantiation for them. Our proposal is general enough to capture the most common forms of mappings between peer ontologies that have been proposed in the literature, and in particular those presented in Section 7.1.

Basically, a peer mapping between ontology $\mathcal{O}^i$ and ontology $\mathcal{O}^j$ is set of assertions between such two ontologies, each relating a view (i.e., a query) specified over $\mathcal{O}^i$ with a view specified over $\mathcal{O}^j$.

The semantic relationship expressed by a mapping assertion depends on the semantic interpretation adopted for each assertion, and on the language adopted for specifying the queries in the assertion. In general, different query languages may be adopted for querying different ontologies.

According to the formalization for P2PDISs we intend to capture here, we assume that the query language used for all mapping views is the language of conjunctive queries.

Then, for each $i, j \in \{1, \ldots, k\}$, the mapping language $\mathcal{L}_M^{ij}$ allows for specifying a peer mapping $\mathcal{M}^{ij}$ from the ontology $\mathcal{O}^i$ to the ontology $\mathcal{O}^j$ as a set of *peer mapping assertions* of the form

$$cq_i \rightsquigarrow cq_j,$$

where cq_j, i.e., the head of the assertion, is a conjunctive query over the ontology $\mathcal{O}^j$, while cq_i, i.e., the tail of the assertion, is a conjunctive query of the same arity as cq_j over the ontology O^i. In other words, the mapping language instantiation above allows for precisely expressing peer mappings for P2PDISs.

7.2.3 Instantiation of the framework for peer ontologies

We finally consider instantiations of the semantic specification of peer ontologies. As already said we provide two different instantiations for capturing the two different formalizations discussed in Section 7.1. We first provide a first-order logic based instantiation, then we provide an epistemic-logic based instantiation.

First-order semantics for systems of peer ontologies. Based on the instantiation of each situated ontology formalism described above, we now instantiate the framework to capture the "classical" approach to providing a semantics to the whole system of peer ontologies. In this approach, one associates to a system $\mathcal{P} = \langle \{\mathcal{O}^i\}_{1 \leq i \leq k}, \{\mathcal{M}^{ij}\}_{1 \leq i,j \leq k} \rangle$ of $\mathcal{F}_P$-peer ontologies, a *single* (open) FOL theory $T_{\mathcal{P}}$, obtained as the disjoint union of the various FOL theories corresponding to the instantiations of each situated ontology formalism $\mathcal{F}_P^i$ discussed above[8] (peer mappings are not considered in $T_{\mathcal{P}}$).

[8]Notice that the instantiation of each $Sem_{\mathcal{E}}^i$ has been given in terms of first-order logic.

By following the approach used for each single situated ontology in the system, we consider a *state E for $\mathcal{P}$*, simply as the (disjoint) union of one local state E^i for each ontology $\mathcal{O}^i$ in $\mathcal{P}$. We call *FOL interpretation of $T_\mathcal{P}$ based on E* an interpretation $\mathcal{I}$ of the FOL theory $T_\mathcal{P}$ such that for each relational symbol s of the environments in the situated ontologies of $\mathcal{P}$, we have that $s^\mathcal{I} = s^E$. Then, the semantic specification for peer ontologies $Sem_P = \langle \mathcal{S}_P, \delta_P \rangle$ is such that,

- $\mathcal{S}_P$ is the set of FOL interpretation of $T_\mathcal{P}$ based on E, and
- for each $\mathcal{I} \in \mathcal{S}_P$, $\delta_P(\mathcal{I}, \mathcal{P}) = \mathsf{true}$ iff
  - each ontology $\mathcal{O}^i$ in $\mathcal{P}$ is satisfied by $\mathcal{I}$, i.e., $\mathcal{I} \in Mod_\mathcal{F}(\mathcal{O}^i)$, and
  - $\mathcal{I}$ is a model of the formula

$$\forall \mathbf{x}.(\exists \mathbf{y}.body_{cq_i}(\mathbf{x}, \mathbf{y}) \supset \exists \mathbf{z}.body_{cq_j}(\mathbf{x}, \mathbf{z}))$$

    for each peer mapping assertion $cq_i \rightsquigarrow cq_j$ belonging to $\bigcup_{i,j=1}^{i,j=k} \mathcal{M}^{ij}$.

  In other words, δ_P is the standard first-order logic applied to a system for peer ontologies $\mathcal{P}$.

Finally, let us consider the query answering service in a system of peer ontologies. Any ontology of the system can be queried by external clients (both users or other ontologies). Let us assume that queries are conjunctive queries over the TBox of the ontology. Given a query q over one ontology $\mathcal{O}^i$ in $\mathcal{P}$ and a state (i.e., a database instance) E for the environment (i.e., a relational database schema) $\mathcal{E}$ (which can be seen as the disjoint union of all environments), we define the *certain answers* $ANS_{fol}(q, i, \mathcal{P}, E)$ to q in $\mathcal{P}$ based on E under FOL semantics, as the set of tuples $\mathbf{t}$ of constants in Σ_o such that for every (FOL) $\mathcal{F}_P$-model $\mathcal{I}$ of $\mathcal{P}$, we have that $\mathbf{t} \in q^\mathcal{I}$ (notice that this notion exactly corresponds to the notion of certain answers given in Section 7.1.2).

Epistemic semantics for systems of peer ontologies. We give now a different instantiation of the semantic specification of peer ontologies. Basically, such new instantiation differs from the previous one for the way in which it interprets peer mapping assertions. In particular, the semantics is not based (as in the previous case) on providing a single FOL interpretation for the whole system: rather, an interpretation structure for the system is now a set of FOL interpretations, which can be seen as the set of possible states of the system. This *possible worlds* semantics corresponds to an *epistemic* interpretation of the mappings [CDGLR04] (analogous to the one described in Section 7.1.4).

As a consequence of this interpretation, we consider each ontology in the system as an independent peer, with its own set of FOL interpretations (again depending on states assigned to the environment). This basically means that we did not provide anymore a single FOL theory for the whole system.

More specifically, given a *state E for $\mathcal{P}$*, such that $E = \{E^1 \cup \ldots \cup E^k\}$, where each E^i is the state of the environment $\mathcal{E}^i$, the semantic specification for peer ontologies $Sem_P = \langle \mathcal{S}_P, \delta_P \rangle$ is now such that,

- $\mathcal{S}_p$ is the set of all $\mathcal{S}_h = \{\mathcal{S}_h^1, \ldots, \mathcal{S}_h^k\}$ such that each $\mathcal{S}_h^i$ is a set of FOL interpretations $\mathcal{I}^i$ for the ontology $\mathcal{O}^i$ based on E^i, i.e., for each relational symbol s of the environment $\mathcal{E}^i$, we have that $s^{\mathcal{I}^i} = s^{E^i}$;
- for each $\mathcal{S}_h \in \mathcal{S}_P$ we have that $\delta_P(\mathcal{S}_h, \mathcal{P}) = \mathsf{true}$ iff
  - each ontology $\mathcal{O}^i$ in $\mathcal{P}$ is satisfied by each $\mathcal{I}^i \in \mathcal{S}_h^i$, and
  - for each peer mapping assertion $cq_i \rightsquigarrow cq_j$ belonging to $\bigcup_{i,j=1}^{i,j=k} \mathcal{M}^{ij}$ and for each tuple $\vec{t}$ of constants of Σ_o, one of the following condition holds
    - there exists $\mathcal{I}^i \in \mathcal{S}_h^i$ such that $cq_i(\vec{t})$ is not satisfied in $\mathcal{I}^i$;
    - $cq_j(\vec{t})$ is satisfied in $\mathcal{I}^j$,for each $\mathcal{I}^j \in \mathcal{S}_h^j$.

It should be easy to see that the above semantics actually corresponds to the multi-modal epistemic semantics used for the epistemic formalization of P2PDISs presented in Section 7.1.4.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[BBL05] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BCN92] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., 1992.

[BGK+02] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. of the 5th Int. Workshop on the Web and Databases (WebDB 2002)*, 2002.

[BH91] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[BH92] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German Workshop on Artificial Intelligence (GWAI'92)*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 1992.

[BM01] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes – W3C recommendation. Technical report, World Wide Web Consortium, May 2001. Available at `http://www.w3.org/TR/xmlschema-2/`.

[BPS94] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language reference – W3C recommendation. Technical report, World Wide Web Consortium, Feb. 2004. Available at `http://www.w3.org/TR/owl-ref/`.

[CDGL+05] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CDGL+06a] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[CDGL+06b] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data management in peer-to-peer data integration systems. In *Global Data Management*. 2006.

[CDGL+06c] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Submitted for publication, 2006.

[CDGLR04] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*, pages 241–251, 2004.

[Che76] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, Mar. 1976.

[Che80] B. F. Chellas. *Modal Logic: An introduction*. Cambridge University Press, 1980.

[CL93] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[FKLS03] E. Franconi, G. Kuper, A. Lopatenko, and L. Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proc. of the VLDB International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003)*, 2003.

[FKMP03] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 207–224, 2003.

[GHVD03] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, pages 48–57, 2003.

[HB91a] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1991. An abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*.

[HB91b] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, 1991.

[HH01] J. Heflin and J. Hendler. A portrait of the Semantic Web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[HIST03] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of the 19th IEEE Int. Conf. on Data Engineering (ICDE 2003)*, pages 505–516, 2003.

[HKS05] I. Horrocks, O. Kutz, and U. Sattler. The irresistible $\mathcal{SRIQ}$. In *Proc. of the Workshop on OWL: Experiences and Directions (OWLED 2005)*, 2005.

[HKS06] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, 2006.

[HMS05] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[HN90] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. Technical Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1990.

[HPSBT05] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL Rules: A proposal and prototype implementation. *J. of Web Semantics*, 3(1):23–40, 2005.

[HPSvH03] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[HS99] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[HS03] I. Horrocks and U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, 2003.

[HS05] I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[HST99] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[Hul88] R. Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–256. Academic Press, 1988.

[Koc02] C. Koch. Query rewriting with symmetric constraints. In *Proc. of the 2nd Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS 2002)*, volume 2284 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2002.

[Len02] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[LL01] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[LR98] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[OCE06] M. M. Ortiz, D. Calvanese, and T. Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, 2006.

[PSHH04] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language semantics and abstract syntax – W3C recommendation. Technical report, World Wide Web Consortium, Feb. 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[PSMB+91] P. F. Patel-Schneider, D. L. McGuinness, R. J. Brachman, L. A. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.

[Sat96] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96)*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer, 1996.

[Sch94] A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[SS89] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89)*, pages 421–431. Morgan Kaufmann, 1989.

[SWM04] M. K. Smith, C. Welty, and D. L. McGuiness. OWL Web Ontology Language guide – W3C recommendation. Technical report, World Wide Web Consortium, Feb. 2004. Available at `http://www.w3.org/TR/owl-guide/`.

[Tob00] S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research*, 12:199–217, 2000.