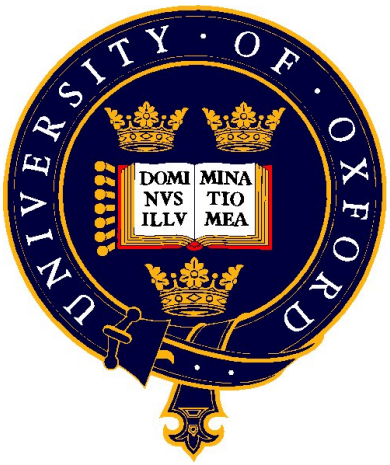
University of Oxford



Representing Graph-based Structures with Logic

by

Despoina Magka

Lincoln College

under joint supervision by Prof. Ian Horrocks and Dr. Boris Motik

Oxford University Computing Laboratory

November 2010

Contents

Chapter 1

Introduction

1.1 General Background and Problem Statement

Knowledge representation and reasoning is a branch of Artificial Intelligence that studies ways to encode knowledge so that inference of new knowledge is done in an automatic and efficient way. In order to achieve this goal, a knowledge representation language and a reasoning procedure need to be designed; the purpose of the reasoning procedure is to manipulate and derive new knowledge from the existing —represented by the language— knowledge. Unfortunately, the more expressive is a language, the less efficient is the associated reasoning algorithm. Very expressive representation languages usually have high (e.g. exponential) complexity reasoning algorithms or fail to have a reasoning procedure because the reasoning problem for them is not decidable. As a consequence, the trade-off between the expressivity of the representation language and the computational cost or existence of the corresponding algorithm is one of the main objects of research.

Complex objects with tree-shaped structure can be described using a variety of knowledge representation formalisms. For instance, Description Logics is a family of logic-based languages with highly optimised reasoning algorithms, that can successfully represent tree-like structures. The tree-model property of description logics, which accounts for their decidability, ensures that the structure of the objects that description logics represent is interpreted by at least one cycle-free model. As a consequence, Description Logics are not able to precisely describe cyclic structures. Nevertheless, there are numerous cases where objects with complex structure that involves cycles (such as chemical molecules that contain atom rings) need to be encoded. Therefore, formalisms that represent and reason about cyclic structures need to be investigated.

Since arbitrarily complex objects with cycles can be abstracted using graphs, it is reasonable to assume that a graph-based knowledge representation formalism is a suitable approach. In order to address the problem

of modeling non-tree structures, we need, initially, to outline the required features of an appropriate solution. Subsequently, we have to investigate existing knowledge representation formalisms and analyse their suitability to the problem according to the requirements. Unless a satisfactory solution is found, a knowledge representation language and an efficient proof procedure need to be designed in accordance to the specified requirements. The development of a suitable graph-based language with a reasoning algorithm as well as the implementation and evaluation of a prototype system based on the language are the main goals of our project.

1.2 Structure of the Report

The current report is structured as follows:

- Chapter 1 provides general background and a high-level description of the problem which we are interested in.
- In Chapter 2, we describe in detail the problem that concerns us by using an example taken from the human anatomy domain. Next, we discuss problems analogous to the initially presented one; at the end of the chapter we summarise the requirements of an adequate to the problem solution.
- In Chapter 3, we explore a number of knowledge representation formalism and identify which are these features that make them an inappropriate to our problem solution.
- In Chapter 4, we explain in more detail two of the key requirements that our solution should conform to; subsequently, we present a high-level language that can be used to represent objects with arbitrarily complex structure. Next, we present the syntax and semantics of a logic-based language that can be used to describe the same objects and discuss the purposes of each language and the relation between them.
- Chapter 5 concludes the report with an account of the completed tasks as well as of the work that we plan to undertake in the future; finally, it provides a time schedule that includes past and planned activities.

Chapter 2

Problem Description and Motivation

We begin the core part of our document with a detailed description of the cycle modeling problem, which is the problem we are interested in. The problem is presented by means of a motivating example, inspired by the human anatomy domain. Subsequently, we refer to a number of problems analogous to the initially presented one. The problems are taken from various and diverse knowledge domains, which shows that the cycle modeling problem is frequently encountered and, thus, of significant importance. At the end of the chapter, we sum up the essential features of what would constitute a solution to the cycle modeling problem.

2.1 The Cycle Modeling Problem

The problem we are concerned with is how to represent cyclic structures and reason about them. In particular, we are interested in designing a logic-based formalism and an inference algorithm which allow the user to (i) encode the (possibly cyclic) structure of a complex object and its properties and (ii) use the reasoning algorithm to derive logical inferences based on the structure of the object and its properties.

Logic-based languages use *axioms* in order to encode information; they also use *models* in order to interpret the meaning of the axioms. A model *satisfies* an axiom when it successfully interprets its meaning. In order to derive an inference from a set of axioms, every model that satisfies the set of axioms should also satisfy the inference. As a consequence, the inferences that a reasoning algorithm derives from a set of axioms, depend on the precision of the models that satisfy the axioms. In our setting, in order to derive inferences which depend on the cyclic structure of the objects, we need to ensure that the defined models faithfully represent the cycles.

We describe the cycle modeling problem in more detail with the help of

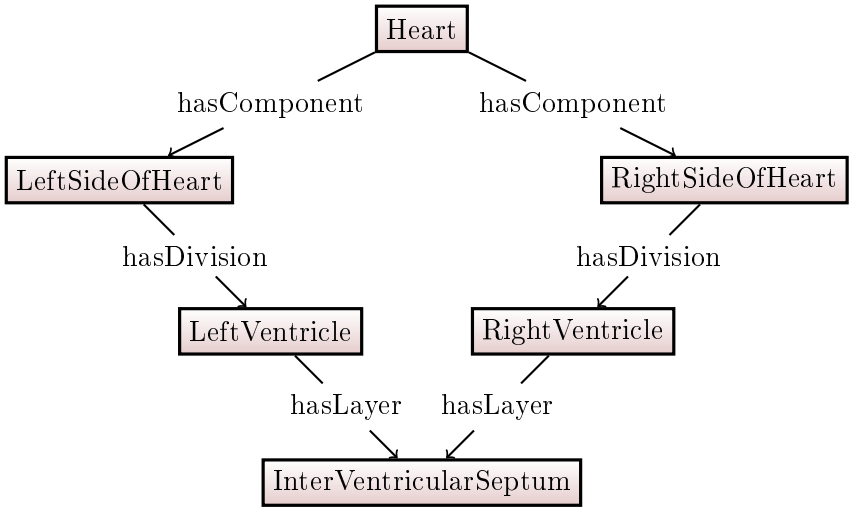


Figure 2.1: Structure of part of the heart

an example taken from the life sciences domain. Assume that we want to encode the structure of (part of) the human heart, which is depicted in Figure 2.1. The human heart has the left and the right side components, which contain the left and the right ventricle, respectively. The two ventricles are separated by a layer, the interventricular septum. The user needs a language that allows him or her to encode the human heart structure with a set of axioms. The models that satisfy these axioms should be consistent with the following facts:

(i) The heart has two components: the left side and the right side.

(ii) The left (right) side has a division which is the left (right) ventricle.

(iii) The left (right) ventricle has a layer which is the interventricular septum.

(iv) The interventricular septum is the same for both the left and the right ventricle.

Moreover, the user might want to specify additional information for the human heart. The language should enable him or her to add axioms, such as the following:

Axiom 1 The left side of the heart has a division with a perforated septum.

Axiom 2 If something has a division with a perforated septum, then it has VSD (ventricular septum defect).

Given the above information, the reasoning algorithm should be able to derive the following two inferences:

(a) The left side of the heart has VSD.

(b) The right side of the heart has VSD (since the right ventricle also has a perforated septum).

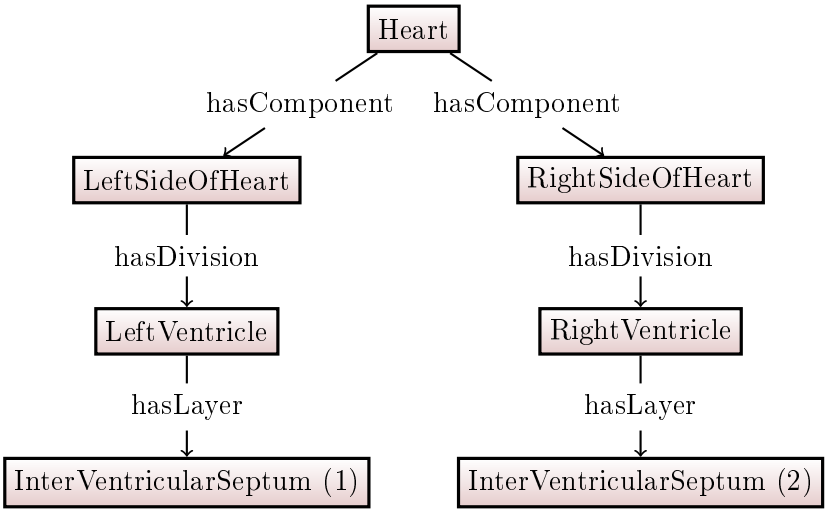


Figure 2.2: Model that satisfies **(i)**–**(iv)**

We now discuss why it is important that the models accurately represent the cyclic structure of the heart. If the language allows only for tree-shaped models (such as e.g. description logics), then the models will be consistent only with facts **(i)-(iii)**; in this case, models such as the one depicted in Figure 2.2 are allowed and inference **(b)** is not derived because there exists at least one model where there are two different septa and only the left one is perforated. On the contrary, if all the models of the language specify that the septum is common to the two sides (fact **(iv)**), then both **(a)** and **(b)** are inferred.

The aforementioned example shows that the models of a language –which are determined by its semantics– influence the inferences that the proof procedure of the language derives. In other words, if the models which do not precisely describe the modeled structure are not excluded, then inferences that are expected to be derived are lost.

The cycle modeling problem can be summed up as the investigation of a formalism with suitable syntax, semantics and reasoning algorithm, such that reasoning tasks similar to the one previously described can be executed. Note that reasoning tasks of this type refer to the structure of the object described (e.g. components of the heart) and not to specific instances of this structure (e.g. the heart of a specific person). Therefore, this is a case of *schema-level* reasoning, where the derivation of new axioms concerns the generic structure (schema) of the object and not instantiations of the object.

2.2 Modeling Cycles in Various Domains

Ontology engineers frequently need to model knowledge domains that involve cycles. There is a variety of different areas with concrete use cases similar to the ones described in Section 2.1. We already analysed a life sciences example in detail. Next, we present a few more practical cases of the same nature, encountered in fields such as engineering, law, chemistry and so on.

2.2.1 Scientific workflows

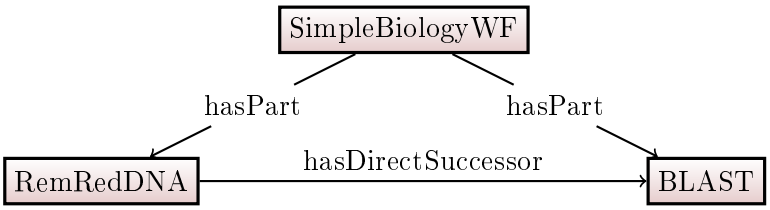


Figure 2.3: Workflow that contains loops

Scientific workflows are directed graphs that describe scientific experiments. The nodes correspond to processes, which form part of the experiments, and the edges indicate the time sequence between the processes. Workflows are difficult to build and, thus, it is important to be able to reuse them. Description logics have been used to formally describe experimental workflows and, thus, to enable workflow reuse [18]. Nevertheless, description logics are not able to handle workflows that contain loops, such as the one depicted in Figure 2.3. Figure 2.3 shows a simple biological workflow which consists of two processes: in the first process the redundant DNA sequences are removed (RemRedDNA) and in the second a sequence-comparing algorithm is executed (BLAST). The problem is analogous to the one described in Section 2.1 and, thus, a solution for the heart anatomy problem would apply to workflow modeling, too.

2.2.2 Engineering

Engineering is an area which, among other things, deals with the manipulation of objects with a highly complicated structure. Various knowledge representation techniques have been adopted by engineers to formally describe these objects.

Consider the problem of representing the design of a product and checking whether a specific implementation complies with certain requirements [19, 20]. The design is very probable to contain cycles, such as the vehicle structure shown in Figure 2.4. SysML [13] is a formal graphical language

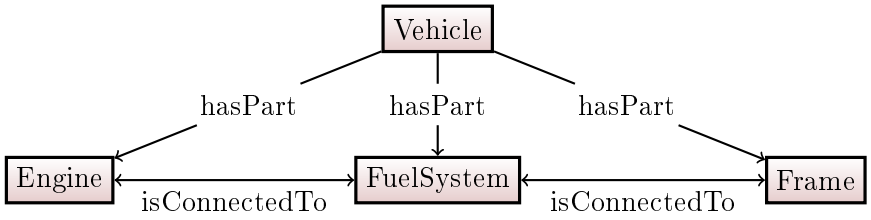


Figure 2.4: Product design with cyclic structure

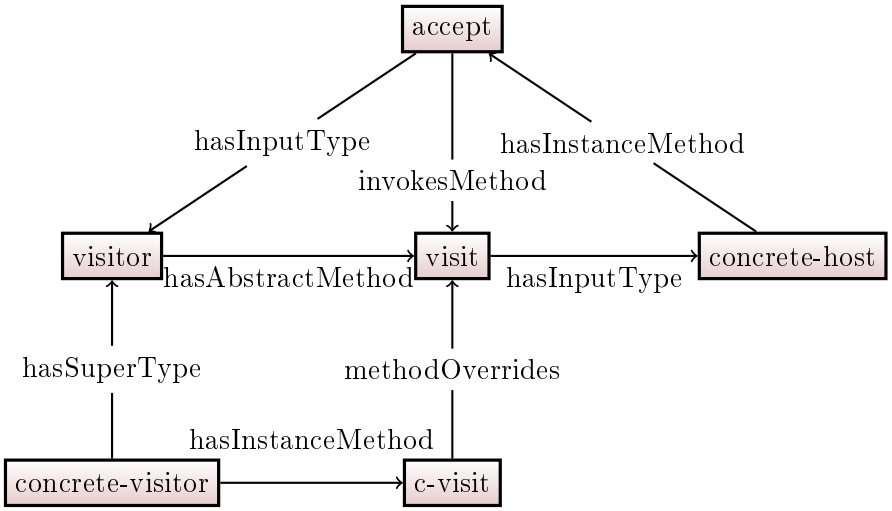


Figure 2.5: The Visitor design pattern

designed for systems engineering; however, SysML is not appropriate for this case as it has no formal semantics. Another problem, encountered in software engineering, is the recognition of design patterns from source code [2]. Some patterns have intricate non-tree structure, such as the Visitor design pattern pattern which is depicted in Figure 2.5. Both problems are, due to their cyclic structure, similar to the problem of Section 2.1

2.2.3 Event recognition

Event recognition deals with deriving events from facts, temporal relations and event-defining rules [49]. It is quite usual for the antecedent facts to be connected in a cycle-like pattern. One of these cases is depicted in Figure 2.6 and is taken from an event recognition application. Figure 2.6 shows when the event of a stressful office day is recognised. The problem is analogous to the one presented in the beginning of the chapter and, thus, a solution to the initially presented problem would apply here, too.

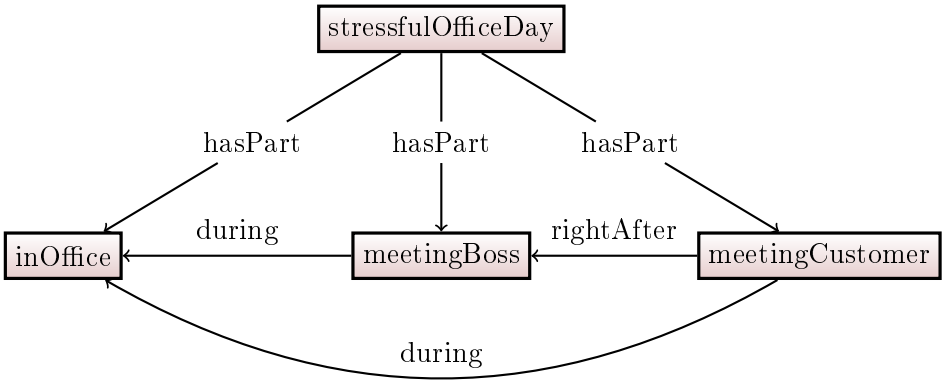


Figure 2.6: Recognizing a stressful office day

2.2.4 Law

Knowledge representation techniques have extensively been used in order to manipulate legal texts. OWL DL ontologies play a central role in this effort [23]. As a consequence, the limitations of OWL DL also restrict the variety of law concepts that can be modeled.

Transaction is an important notion in law because it encodes the pattern of exchange, which often characterizes the actions of people as e.g. it is the case for a sales transaction or a contract [25, 24]. A transaction has two participants: the first one provides an object which is received by the second, whilst the second one provides in exchange another object which is received by the first. A similar structure corresponds to the complementarity of rights and duties. Figure 2.7 illustrates the diamond-shaped structure of transaction. In order to successfully model the structure of the transaction, the actor of the first transfer and the recipient of the second transfer should be the same (similarly for the other agent). OWL DL fails to precisely describe this identification, which is similar to the one encountered in the heart anatomy example.

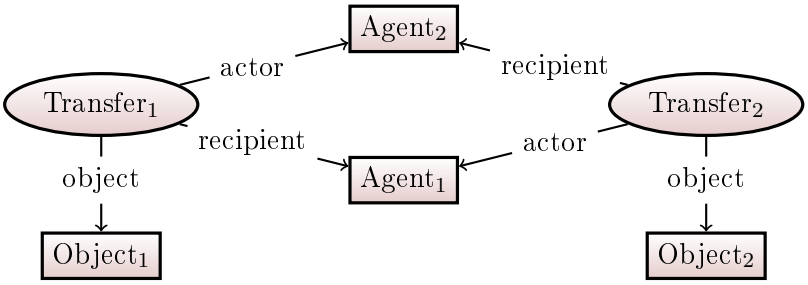


Figure 2.7: Structure of a transaction

Another case where OWL ontologies have been employed in the area of law is the task of legal assessment [46]. The system described accepts as input a set of norms and a case description and, after performing Description Logics reasoning, replies whether the case of the input is allowed or prohibited. Van de Ven et al. [46] present a worked-out example which is based on an ontology that models university library regulations. The system described covers a wide range of use cases. However, there are still situations that cannot be described: if we want to model the case where a student checks out a book belonging to a course *he* or *she* is enrolled in, the problem of identification prevents us from doing so. This is an instance of the cycle modeling problem described in Section 2.1.

2.2.5 Chemistry

In the same sense that knowledge representation techniques have been used to model the life sciences domain, they have also been used to describe chemical information. Various OWL ontologies have been built for the representation and classification of chemical molecules [30, 22].

A key reasoning service that a KR system about chemical ontologies needs to offer is the recognition –and, thus, classification– of molecules in the basis of the *functional groups* they contain. A functional group that a molecule contains is a set of specific atoms appropriately connected with each other; functional groups are a powerful tool because they define how molecules interact with each other. As a consequence, it is crucial for a KR language to be able to recognise functional groups. However, as Figure 2.8 suggests, functional groups may contain rings in their structure and since OWL can concisely represent only cycle-free structures it is not a suitable formalism [48]. Notably, the capability to describe rings seems to be the only expressive feature not satisfied by OWL 1.1 (OWL 2) in the list of features required to classify compounds from functional groups. The problem is similar to that of representing the cyclic structure of human heart.

ChEBI [9] is a database and ontology that describes chemical entities of biological interest. ChEBI currently contains nearly 22,500 entities;[1] it is estimated that the number of entities which are considered "biologically interesting" can reach 1,000,000. Given that the majority of entities is manually inserted in the ontology at a rate of around 1,500 per person per year, there is a lot of potential for the ontology to grow if processes become automatic [26]. A key service that ChEBI should offer is searching for chemical structures. Furthermore, an essential requirement for searching is the ability to represent and reason about circular structures. As a consequence, the development of a logic language that can handle such structures is significant for the ChEBI ontology.

[1] http://www.ebi.ac.uk/chebi/statisticsForward.do

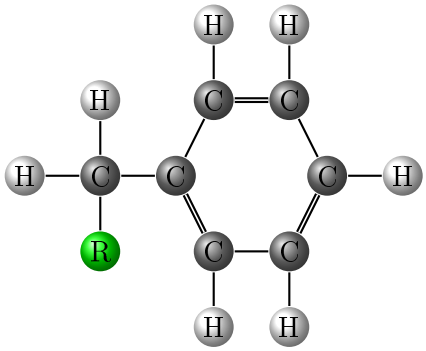


Figure 2.8: The benzyl functional group.

2.3 Key Requirements of the Solution

In order to be able to resolve the problem described in Section 2.1, as well as the similar problems of Section 2.2, we need to design a suitable graph-based language, where complex structures are naturally represented by graphs. We also need to ensure that the suggested formalism complies with the following essential requirements:

- The language should admit a sound, complete and terminating proof procedure, that is the main reasoning problem for the language (e.g. satisfiability) should be *decidable*. Additionally, the decision procedure should be efficient enough, so that implementation is feasible.
- The language should allow for schema-level reasoning, that is for derivation of axioms that refer to the structure of the objects and not to instances of their structure.
- The language should guarantee that concepts which are described by (possibly cyclic) structures are recognised (e.g. the benzyl functional group).
- Finally, it is a requisite for the language to permit the composition of several graphs to a single graph (e.g. the circulatory system consists of the heart, veins, arteries and so on). The language should ensure that a graph described as a composition of smaller graphs is logically equivalent with a graph described by the nodes and edges that the smaller graphs consist of and by some additional edges.

Chapter 3

KR Formalisms Surveyed

In the current Chapter, we investigate a number of knowledge representation (KR) languages as a solution to the cycle modeling problems described in Chapter 2. We provide a high-level description of the features of each language and examine their suitability as a solution to the cycle modeling problem. Section 3.1 discusses monotonic languages that employ the open-world assumption, whereas Section 3.2 deals with non-monotonic formalisms. Subsequently, Section 3.3 considers languages that combine monotonic and non-monotonic characteristics and, finally, Section 3.4 investigates query languages.

3.1 Monotonic Languages

A logic language is monotone, when, given a set of axioms A and the set of derived axioms $D(A)$, extending A to A' ($A \subseteq A'$) results to the equally many or more derived axioms $D(A) \subseteq D(A')$. In the current section, we investigate several prominent examples of monotonic KR formalisms which are relevant to the previously described problem. For every formalism, we examine whether it is appropriate for the aforementioned problem and if not we point out the drawbacks that cause it to be inappropriate.

3.1.1 First-order Logic

First-order logic [12] is one of the most well-known formal logic systems; it is of great importance to the foundations of mathematics and has inspired numerous KR formalisms. Syntactically, first-order logic allows for variables, constants, predicates, functions as well as for formulas built using existential and universal quantifiers, conjunction, disjunction and negation. The semantics of first-order logic is set-theoretic. The definition of a model requires the definition of a non-empty set, the *domain*; constants, predicates and functions are interpreted through an *interpretation function*, which also defines

satisfaction of first-order formulas. A domain combined with an interpretation function form a model, which is an important structure as in many cases it reflects properties of the formulas it satisfies. In first-order logic, quantified variables range over terms and not over predicates or functions.

Proof procedures for first-order logic decide whether a set of first-order formulas is satisfiable –that is, whether a model that satisfies the formulas exist. The two predominant proof procedures are tableau and resolution; both are sound and complete. Given a set of formulas, tableaux algorithms try to construct a model that satisfies them, while resolution algorithms try to derive a contradiction, which implies that no model of the formulas exist. The fact that infinitely many terms can be constructed with the use of function symbols is one of the reasons why tableau and resolution algorithms are non-terminating. In fact, there exists an algorithm that given a formula, outputs "yes" if the formula is valid, but "no" or does not give an answer at all (infinitely runs) if the formula is not valid. Given that satisfiability can be reduced to validity, satisfiability of first-order formulas is a semidecidable problem and, thus, first-order logic does not meet the decidability requirement mentioned in Section 2.3. As a consequence, it is not an appropriate formalism for the cycle modeling problem.

3.1.2 Common Logic (CL)

Common Logic (CL) defines a family of logic languages, whose purpose is to act as a medium of transmitting logical content between computer-based agents [27]. CL has an abstract signature-free syntax and allows for higher-order constructions, such as expressions where quantifiers range over relations or functions. Moreover, CL has a model theoretic first-order semantics which is appropriately adapted according to the presence or absence of higher-order expressions.

It is beyond the scope of CL to specify inference algorithms that check consistency of or derive new CL expressions. Consequently, CL is not a suitable solution for problems, such as modeling the anatomy of the human heart.

3.1.3 HiLog

HiLog is a logical formalism that syntactically subsumes first-order logic and allows for higher-order logic expressions [8]. The syntax of HiLog permits the use of a symbol as a constant, function or predicate symbol with non-fixed arity in the same expression. E.g. in $p(a,b) \wedge q(p(a),b) \rightarrow q(p,a,b)$, the symbol p is used as a binary predicate, an one-argument function and a constant, while q is used as both a binary and ternary predicate. In spite of that, the semantics of HiLog is not the same as the semantics of second-order logic; in HiLog semantics two predicates are equal only when their symbols

are explicitly made equal (e.g. $p = q$), whereas in second-order semantics two predicates are equal if they are interpreted by the same set of object tuples.

HiLog admits a sound and complete proof procedure. It is proved that all formulae produced by the HiLog syntax have an equisatisfiable translation to first-order logic. However, not all first-order formulae which are satisfiable under first-order semantics are necessarily satisfiable under HiLog semantics. Nonetheless, the equality-free first-order formulae enjoy the property of being valid under first-order semantics iff they are valid under HiLog semantics.

The higher order expressivity of HiLog enables HiLog to address the representational aspects of the cycle modeling problem. In particular, in Section 4.3 we present a HiLog-style language which satisfies the concept recognition and graph composition requirements. Similarly to first-order logic, the validity problem for HiLog expressions is not decidable. As a consequence, HiLog is not an appropriate solution for the discussed cycle modeling problem.

3.1.4 Description Logics (DL)

Description Logics [4, 5] (DL) are a family of logic-based languages used for the formulation and manipulation of axioms that model knowledge domains. The DL syntax defines axioms with the help of *concepts*, *roles* and *individuals*. Individuals are similar to constants—they are names for elements of the domain. Concepts, like unary predicates, describe common characteristics shared by a set of individuals. Roles, in the same way as binary predicates, describe links between pairs of individuals. The expressivity of a Description Logic is determined by the constructors that are available in the logic, such as conjunction, negation and existential (or universal) restrictions. DL axioms are (usually) of two kinds: the TBox axioms, which form constraints about concepts and roles, and the ABox axioms, where facts about individuals are stated. DLs have a first-order logic semantics and, thus, employ the open-world assumption.

DL languages have terminating, sound and complete inference algorithms; for expressive logics the algorithms are of exponential complexity but behave well in practice because they are highly optimised. Decidability of DLs is ensured by the tree model property, according to which, every set of satisfiable axioms has at least one tree-shaped model.

We now show how the tree-model property affects the ability of DLs to precisely describe cyclic structures. Several biomedical DL ontologies have been developed [44, 45], such as Galen[1] and FMA.[2] DL ontologies consist of DL axioms, which can be used e.g. to model the anatomy of human organs

[1] http://www.opengalen.org/

[2] http://sig.biostr.washington.edu/projects/fm/index.html

[37], such as (a part of) the human heart (Figure 3.1) which was examined in Section2.1.

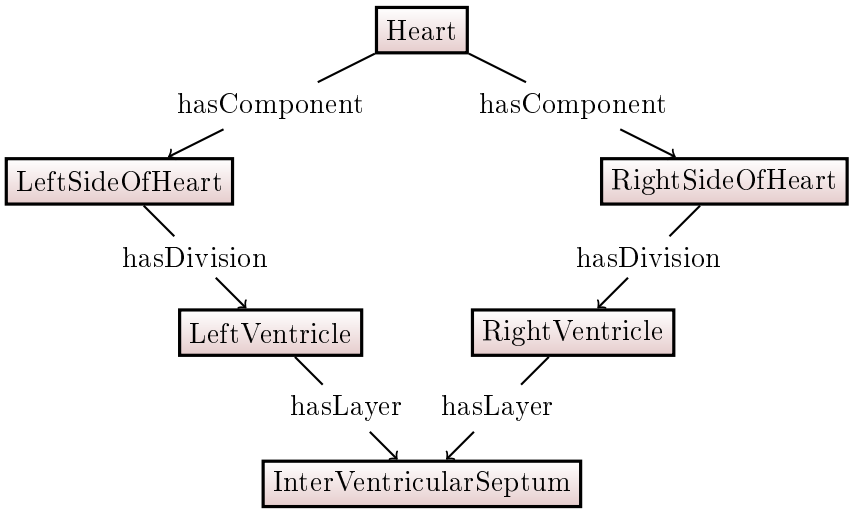


Figure 3.1: Structure of part of the heart

The following DL axioms could be used to describe the structure of the human heart:

$$Heart \sqsubseteq \exists hasComponent.LeftSideOfHeart \tag{3.1}$$

$$Heart \sqsubseteq \exists hasComponent.RightSideOfHeart \tag{3.2}$$

$$LeftSideOfHeart \sqsubseteq \exists hasDivision.LeftVentricle \tag{3.3}$$

$$RightSideOfHeart \sqsubseteq \exists hasDivision.RightVentricle \tag{3.4}$$

$$LeftVentricle \sqsubseteq \exists hasLayer.IntraventricularSeptum \tag{3.5}$$

$$RightVentricle \sqsubseteq \exists hasLayer.IntraventricularSeptum \tag{3.6}$$

If we view Figure 3.1 as a DL model (by mapping appropriately nodes to objects, node labels to concept interpretations and edge labels to role interpretations), we note that the model satisfies the axioms (3.1)–(3.6). However, the model that corresponds to Figure 3.2, satisfies the axioms (3.1)–(3.6), too. The second model is more general than the first one, as it represents the least amount of information derivable from the DL axioms (e.g. existential constraints). In the contrary, the first model requires the septum of left and

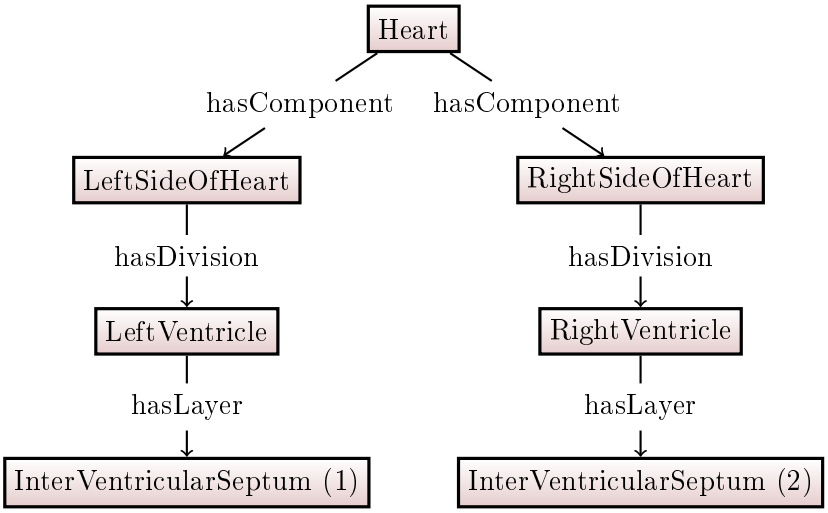


Figure 3.2: Model that satisfies axioms (3.1)–(3.6)

right ventricle to be the same, which might be the intention of the modeller but is not stated by the DL axioms. As we see next, the two models also satisfy different DL axioms. Let us assume that we add axioms (3.7) and (3.8). Axiom (3.7) says that the left side of the heart has a division with a perforated septum and Axiom (3.8) says that, if something has a division with a perforated septum, then it has ventricular septum defect (VSD).

$$\begin{aligned} &LeftSideOfHeart \\ &\sqsubseteq \exists hasDivision.\exists hasLayer.PerforatedSeptum \end{aligned} \tag{3.7}$$

$$\begin{aligned} &\exists hasDivision.\exists hasLayer.PerforatedSeptum \\ &\sqsubseteq hasVSD \end{aligned} \tag{3.8}$$

Suppose that $\mathcal{I}_1$ is the model corresponding to Figure 3.1, $\mathcal{I}_2$ the model corresponding to Figure 3.2 and K includes axioms (3.7)–(3.8). Let A be the axiom $RightSideOfHeart \sqsubseteq hasVSD$. We have that if $\mathcal{I}_1 \models K$, then $\mathcal{I}_1 \models A$, because there is only one septum, which causes the right side of the heart to have VSD, too. Nevertheless, if $\mathcal{I}_2 \models K$, then $\mathcal{I}_2 \not\models A$, because there are now two different septa in the model and only the left one is perforated. As a consequence, $K \not\models A$, which is contrary to what we would expect: if the left side of the heart has a division that has a perforated septum, then the right side of the heart has a division with a perforated septum too (as the septum is common to the two sides) and, thus, has VSD. DL axioms fail to express that the two septa are the same, due to the tree-model property [47], which requires at least one of the models satisfying the axioms (3.1)–(3.8) to be tree-shaped.

One could use ABox axioms to specify the cyclic structure of the heart. This is not an appropriate solution though. In this case we would define an instance of the heart –that is, one particular heart– instead of encoding the structure of the heart and instantiating it, e.g. for each patient, separately.

As we saw, description logics can represent only cycle-free structures in a sufficiently precise way; so, they are not a suitable solution for the cycle modeling problem.

3.1.5 DL-Safe rules

DL-safe rules is a monotonic and decidable rule formalism that extends description logics. DL-safe rules can be used to describe arbitrary, and not necessarily tree-shaped, structures. A hybrid language has been presented that combines DL axioms and DL-safe rules in order to perform reasoning over knowledge bases that contain both [41].

The syntactic form of a DL-safe rule is:

$$A_1 \vee \ldots \vee A_m \leftarrow B_1 \wedge \ldots \wedge B_n$$

where A_1, ..., A_m, B_1, ..., B_n are atoms of the form $P(s_1, \ldots, s_k)$ (P is a predicate and s_i are terms). A DL-atom is an atom $P(s_1, \ldots, s_k)$, where P is a concept or a simple role that occurs in a DL axiom. The DL-safety restriction requires every variable of the rule to occur in at least one non-DL-atom in the body of the rule. DL-safety is dealt with by introducing a trivial non-DL unary predicate, $O(x)$, and add empty-bodied rules (facts) for all known individuals of the knowledge base (e.g. $O(a)$ for a). DL-safe rules are interpreted under first-order semantics. A query-answering resolution-based algorithm has been developed for knowledge bases with DL $\mathcal{SHIQ}$ axioms and DL-safe rules that runs in deterministic exponential time.

In order to secure decidability for description logics and function-free rules different restrictions need to be imposed in each of the two cases; as a consequence if we want to have a knowledge base that combines description logics axioms and function-free rules we should impose appropriate restrictions that guarantee decidability for both formalisms. In the case of description logics, decidability is ensured by restricting our attention to tree-like finite models; in the other case, rules are grounded in all possible ways with the constants of the knowledge base and propositional reasoning is performed, which is decidable. The grounding accounts for the DL-safety restriction, which limits the reasoning to the known objects. Consequently, no inferences are derived for unknown or implied by existential quantifiers objects and the conceptual reasoning requirement is not met. We demonstrate this limitation by means of an example. Assume that we have a knowledge base that contains the DL-axiom (3.9) and the DL-safe rule (3.10):

$$NieceWithAunt \sqsubseteq Female \sqcap \exists hasParent.\exists hasSister.\top \qquad (3.9)$$

$$hasAunt(x,y) \leftarrow hasParent(x,z) \wedge hasSister(z,y) \wedge O(x) \wedge O(y) \wedge O(z) \tag{3.10}$$

Assume also that the ABox contains the following assertions:

$$Niece(claire), hasParent(claire, bob), hasSister(bob, alice)$$
$$O(claire), O(bob), O(alice)$$

The DL-safe rule manages to encode the cyclic structure which underlies auntness and, thus, derives $hasAunt(claire, alice)$. Nevertheless, due to the predicates $O(x)$, $O(y)$ and $O(z)$ the rule is applicable only to the named individuals (*claire*, *bob*, *alice*) and not to the ones implied by existential restrictions. Therefore, inspite of the fact that every niece with an aunt n has a parent p with a female sibling s, the axiom $NieceWithAunt \sqsubseteq \exists hasAunt.\top$ is not derived. As a consequence, DL-safe rules do not support schema-level reasoning and, thus, do not match the requirements of our problem.

3.1.6 Description Graphs (DG)

Description graphs (DG) are knowledge modeling constructs, which in combination with description logics and rules can provide a faithful schema-level description of structured objects, whose components are arbitrarily connected [35, 34, 33, 38, 36]. Additionally, when suitable syntactic restrictions are applied to the logic, the reasoning problems for description graphs become decidable.

Description graphs are directed graphs, whose nodes are labeled by unary predicates (concepts) and edges by binary predicates (roles). The knowledge bases that contain description graphs, may also contain description logic axioms and rules, i.e. function-free implications. The interactions between DGs, DL axioms and rules causes the satisfiability problem to be very easily undecidable. In order to regain decidability, syntactic restrictions are imposed; for instance, the roles used in the rules and description graphs are different from the roles used in the description logic axioms. Additionally, an acyclicity condition for the graphs is required in order to ensure that every complex object, which is modeled by a DG, is described up to a certain degree of granularity; the acyclicity condition prevents the language from encoding infinite hierarchical structures with graphs. Moreover, it is proved that for a knowledge base that contains DGs, DL axioms and rules, when the DL used is $\mathcal{SHIQ}^+$ or $\mathcal{SHOQ}^+$ and the appropriate syntactic restrictions apply, the satisfiability problem is NEXPTIME-complete. In addition to that, a prototypical implementation of the formalism reveals that it behaves well in practice, as there is no significant deviation of the classification time for OWL ontologies; furthermore, domain experts state that the models constructed for the description graphs formalism correspond better to intuition than models that satisfy DL axioms.

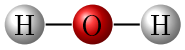


Figure 3.3: Molecule of water.

The formalism described manages to address the lost inference problem of the human heart example: the expected axioms are now entailed as all the non-intended models that have a tree-like structure are excluded. Nevertheless, the description graphs formalism does not satisfy the concept recognition requirement outlined at the end of Chapter 2. For instance, assume that the water molecule which is depicted in Figure 3.3 is encoded by a description graph. Assume also that the ABox contains the following assertions:

$$\begin{array}{lll} hasAtom(m,o) & Oxygen(o) & \\ hasAtom(m,h_1) & Hydrogen(h_1) & bond(o,h_1) \\ hasAtom(m,h_2) & Hydrogen(h_2) & bond(o,h_2) \end{array}$$

Unless a rule of the form:

$$\begin{aligned} &Hydrogen(x_1) \wedge Hydrogen(x_2) \wedge Oxygen(x_3) \wedge bond(x_3,x_1) \\ &\wedge bond(x_3,x_2) \wedge \bigwedge_{1 \leq i \leq 3} hasAtom(x_3,x_i) \rightarrow Water(x_3) \end{aligned} \tag{3.11}$$

is added to the knowledge base m is not recognised as a water molecule and $Water(m)$ is not inferred. Moreover, if rule (3.11) is included to the knowledge base and the assertions $hasAtom(m,h_3)$, $Hydrogen(h_3)$ and $bond(o,h_3)$ are added (where $h_3 \not\approx h_1$, $h_3 \not\approx h_2$), then $Water(m)$ is still derived, which should not happen since m has now additional structure. The example shows why the description graphs formalism fails to meet the concept recognition requirement; as a consequence it is an inappropriate for our setting solution.

3.2 Non-monotonic Languages

For monotonic languages adding new information to some existing knowledge never cancels any of the conclusions already drawn. Nevertheless, this is not usually the case for real world scenarios, where the add of new facts might cause some of the derived facts to become false. As a consequence, various non-monotonic KR formalisms have been developed. In order to investigate whether non-monotonic formalisms are useful for our case, we explore two of them: logic programming and datalog. We give a brief overview of the features of the languages and investigate their suitability for the cycle modeling problem.

3.2.1 Logic Programming (LP)

Logic programming (LP) emerged as a new declarative programming language with a syntax that strongly resembles first-order logic. Unlike first-order logic, LP is a non-monotonic language and, thus, can be used to faithfully represent facts of non-monotonic worlds.

We discuss the most well-known LP formalisms; we begin with the simplest and move on to the more complicated ones. We adopt the same logic programs naming as Baral and Gelfond do [6] and use some standard first-order logic terminology. A *General Logic* program P is a set of rules of the form:

$$A_0 \leftarrow A_1 \wedge \ldots \wedge A_m \wedge not\ A_{m+1} \wedge \ldots \wedge not\ A_n \tag{3.12}$$

where A_i, $0 \leq i \leq n$, are atoms of the form $p(t_1, \ldots, t_k)$, p is a predicate of arity n and $t_1, \ldots, t_k$ are terms. We call the part on the right of the arrow the *body* of the rule and the part on the left the *head* of the rule. The connective *not*, which is used in (3.12), is a form of negation which is known as negation-as-failure and is different from the negation ($\neg$) of first-order logic. Intuitively, *not* is used to state that a fact is false, when the fact is not entailed (failed to be derived) by the program. Formally, the use of *not* is defined by the *stable model semantics* [15, 3], which are the semantics of the general logic programs. The stable model of a logic program P specifies which are the correct answers to queries over P. The intuition behind stable models of a logic program is to identify a set of facts that a rational agent assumes that hold. For the simplified case where A_i, $0 \leq i \leq n$, are propositional variables, a stable model of a logic program P is the minimal w.r.t set inclusion model M of P^M, where P^M is P after substituting every A_i, $m+1 \leq i \leq n$, with its value in M. A logic program can have one, many or none stable models. For instance the logic program that consists of the clauses $s \leftarrow p$, p and $notq$ has one stable model ($\{p : True, s : True, q : False\}$), the program with $p \leftarrow notq$ and $q \leftarrow notp$ has two ($\{p : True, q : False\}$ and $\{q : True, p : False\}$) and the program with $p \leftarrow notp$ has none. The decision problem for general logic programs under stable model semantics is coNP-complete [32].

An important property of logic programs, which is related to its computational properties, is whether they are stratified or not. A logic program P with rules of the form (3.12) is stratified if there is a function $s : Pr \rightarrow \mathbb{N}$, where Pr is the set of predicate symbols of P, such that for each rule (i) $s(A_0) \geq s(A_i)$, for $i = 1, \ldots, m$ and (ii) $s(A_0) > s(A_i)$, for $i = m+1, \ldots, n$. Stratified programs have a unique (stable) model. The complexity of evaluating queries over stratified programs is lower than over non-stratified programs.

An *extended logic* program P is a set of rules of the form:

$$L_0 \leftarrow L_1 \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \tag{3.13}$$

where L_i, $0 \leq i \leq n$, are of the form $p(t_1, \ldots, t_k)$ or $\neg p(t_1, \ldots, t_k)$. Extended logic programs may contain two forms of negation: (i) negation-as-failure (denoted by *not*) which is the same as for general logic programs and (ii) strong negation (denoted by $\neg$), where $\neg A$ holds when it is entailed by the program. The satisfiability of extended logic programs is defined through answer set semantics, which appropriately extend stable model semantics[14]. The existence of two kinds of negation enables an extended logic program to adopt for each predicate q the closed world assumption by including a rule of the form $\neg q \leftarrow not\ q$; if no rule is added the open-world assumption is adopted by default. As a consequence, an extended logic program can choose between the closed and the open world assumption for each predicate, depending on whether it has complete or incomplete information about it, respectively [31].

Logic programming with answer set semantics is known as Answer Set Programming (ASP) and, among other things, is particularly efficient in solving NP-hard search problems [11]. In order to solve a search problem, the problem is initially encoded with program clauses. Subsequently, the program is grounded, that is the variables are replaced by the constants of the program, in all possible ways. Next, the search process is reduced to finding the stable models of the program. Finally, the constraints of the program (which are clauses with empty heads) further restrict the set of models and the solution is the remaining stable models.

Logic programs can be further extended by allowing a disjunction of literals in the head of the rule. A *disjunctive logic* program P is a set of rules of the form:

$$L_0 \vee \ldots \vee L_k \leftarrow L_{k+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n$$

where L_i, $0 \leq i \leq n$, are literals defined as before. A new semantics of disjunctive logic programs has been defined that also takes into account the disjunction. The decision problem for disjunctive logic programs under total disjunctive stable model semantics is Π_2^P-complete ($co - NP^{NP^P}$-complete) [43].

As far as our problem is concerned (Section 2.1), Logic Programming is not a suitable formalism, because, due to the grounding phase, it performs reasoning only over constants –that is, known objects of the knowledge base– which is not sufficient for schema-level reasoning.

3.2.2 Datalog

Datalog is a rule language, which is strongly influenced by logic programming, but designed to be used in deductive databases applications [1]. In this context, the predicates correspond to relation names, the rules with empty body to database facts and the constants to data that populate the database.

In its simplest form, the syntax of datalog is a function-free version of general logic programs syntax (3.12) without negation ($n = m$) and where all variables of A_0 occur in some A_i, $1 \leq i \leq m$. Various semantics have been suggested for this version of datalog, such as model-theoretic semantics (minimal model w.r.t. set inclusion), fixpoint semantics (use of fixpoint operator) or proof-theoretic semantics (involve SLD resolution). The decision problem for this case is EXPTIME-complete w.r.t. program complexity.

If we allow for negation in the body of the rule (that is $n > m$) and in the body or the head of the rule, datalog is extended to datalog$^{\neg}$ and datalog$^{\neg\neg}$, respectively. Datalog$^{\neg}$ and datalog$^{\neg\neg}$ are interpreted under stratified and well-founded semantics correspondingly. The decision problem for the stratified semantics is EXPTIME-complete w.r.t program complexity.

If the head of the rule is permitted to contain a disjunction of atoms, the language is called disjunctive datalog. Several semantics have been proposed for disjunctive datalog, such as perfect model semantics and appropriately extended versions of minimal model and stable model semantics. The decision problem for this case is proved to be NEXPTIME$^{\text{NP}}$-complete [10], w.r.t. program complexity.

Various other versions of datalog have been suggested, which are less or more expressive than the ones described above. A wide range of optimizations has also been investigated for datalog reasoning algorithms.

Although the datalog is substantially based on logic programming, there exist some differences between the two formalisms. Unlike LP, datalog does not allow function symbols and requires all the variables of the head to appear in the body of the rule. Due to the absence of functions, datalog programs always have finite models as opposed to LP models that may be infinite. Moreover, in datalog programs —and not in LP—- variables that occur in the head of the rule have to occur in the body of the rule. As a consequence reasoning is restricted only to explicitly named objects of the knowledge base and has the same limitations as for the case of DL-safe rules. As a consequence, datalog does not satisfy the schema reasoning requirement outlined at the end of Chapter 2 and cannot be used to satisfyingly solve the cycle modeling problem.

3.3 Hybrid Approaches

Subsequently, we present some KR formalisms which combine both monotonic and non-monotonic features. Similarly to Sections 3.1 and 3.2, we overview the features of the language and examine whether the language suits the problem discussed in Chapter 2.

3.3.1 Knowledge Interchange Format (KIF)

Knowledge Interchange Format (KIF) was created to facilitate the interchange of knowledge between computer systems that do not share a common language [16]. KIF was developed as part of the DARPA Knowledge Sharing Effort project, whose objective was to design a computer interlingua for communication and, thus, knowledge sharing.

KIF is equipped with a wide range of expressive features; all expressions that are syntactically conformant to first-order logic can be encoded using KIF. Additionally, KIF allows for quantification over logical expressions, functions and relations. It is also acceptable in KIF to use the same name for individuals, functions or relations of different arity. In addition to that, it offers support for numbers, non-monotonic rules and the modeling of beliefs.

Since all FOL expressions can be encoded in KIF and, as we already discussed, FOL is a semidecidable formalism, it is not possible to suggest a terminating reasoning algorithm for KIF. The lack of reasoning services makes KIF an inappropriate solution for the cycle modeling problem.

3.3.2 F-Logic

F-Logic was devised with the objective of providing a theoretically robust framework for object-oriented databases [29]. F-Logic semantics is defined in such a way that basic object-oriented notions are incorporated in the language, such as inheritance, typing, methods, encapsulation and description of complex objects. Additionally, it is capable of representing higher-order concepts but without adopting higher-order semantics: this means that variables do not range over domains of relations or domains of functions but they range over "special" objects of the domains (*intensions*), which have relations and functions attached to them.

As far as reasoning is concerned, the monotonic part of the language has a sound and complete proof procedure and it, also, has a direct translation to first-order logic [28]. There is also a non-monotonic part of the language which admits a sound, albeit an incomplete, reasoning algorithm. F-Logic can be used to adequately define, query and manipulate database schemas. Apart from that, it finds applications in the area of Artificial Intelligence and specifically in frame-based languages, from which it also derives its name.

F-Logic is generally undecidable. The syntax of an implemented (and, thus, decidable) version of F-Logic [42] requires that every variable in the head of the rule must also occur in a positive atom in the body of the rule. This restriction prevents us from reasoning about unknown objects, as we cannot e.g. express the fact that every patient has an (unknown) heart. As a consequence, F-Logic is not an appropriate formalism for the considered problem.

3.3.3 Minimal Knowledge Negation Failure (MKNF)

Logic programming has been combined with description logics to a hybrid logic, Minimal Knowledge Negation Failure (MKNF) [39], which is a function-free language that inherits characteristics from both languages. Description Logics adopt the open-world assumption: a fact which is not explicitly negated in the knowledge base is not considered to be false. The objective of MKNF is to create a logic that handles cases where both, closed-world querying and modeling with description logics, are required. In order to address this problem, MKNF seamlessly integrates open and closed world reasoning in a single language. Additionally, MKNF has a computational advantage, as the data complexity of MKNF does not exceed the corresponding complexity of logic programming used separately [40]. Nevertheless, it cannot be used to satisfyingly solve the cycle modeling problem, as MKNF uses rules that are applicable only to named objects of the ontology and not to unknown objects, e.g. objects implied by an existential quantifier.

3.4 Query Languages

In the current section, we examine two query languages and their appropriateness for the cycle modeling problem. The reason for focusing on query languages is that one of them (nRQL) has been used to tackle one of the problems presented in Chapter 2 [49].

3.4.1 EQL-Lite

EQL-Lite [7] is a query language for description logics which allows to formulate first-order logic queries over DL knowledge bases. Unlike databases, query answering with first-order logic queries is a difficult task due to incomplete information, which is a consequence of the open-world assumption adopted by description logics.

Calvanese et al. [7] suggest the use of a nonmonotonic epistemic FOL query language, EQL-Lite, in order to address the FOL queries problem. In particular, a query in EQL-Lite($\mathcal{Q}$) is a formula of first-order logic with equality, where atoms are of the form $\mathbf{K}\rho$ and ρ is a query in language $\mathcal{Q}$. Intuitively, $\mathbf{K}\rho$ can be perceived as "it is known that ρ". With the appropriate use of the modality operator $\mathbf{K}$, queries can be evaluated under the closed-world assumption. It is proved that an EQL-Lite($\mathcal{Q}$) query q over a DL knowledge base Σ retrieves the same answers as when q_{FOL} is evaluated over Σ_{FOL}, where q_{FOL} and Σ_{FOL} are appropriate FOL translations of q and Σ. Moreover, EQL-Lite($\mathcal{Q}$) does not insert additional computational cost: queries written in EQL-Lite($\mathcal{Q}$) can be evaluated with the same data complexity as queries expressed in $\mathcal{Q}$. As a consequence, EQL-Lite($\mathcal{Q}$) provides a powerful mechanism for querying DL knowledge bases. Since EQL-Lite($\mathcal{Q}$)

is used to query DL knowledge bases and DLs are not suitable for the representation of cyclic structures (Section 3.1.4), EQL-Lite($\mathcal{Q}$) cannot be used to solve the problem presented in Chapter 2.

3.4.2 nRQL Queries

New Racer Query Language (nRQL) is a practical-oriented query language designed as a response to the request by the users of DL reasoner RacerPro [17] for more expressive queries[50, 21]. nRQL queries allow for negative literals, negation-as-failure and formulas built by Boolean connectives, in the body of the queries. The allowance of negation-as-failure makes nRQL a non-monotonic DL query language. nRQL queries may refer to unnamed ABox individuals and, thus, query evaluation is not limited to the set of known objects.

Queries are different from rules, in the sense that they are not part of the knowledge base but they are asked to a knowledge base in order to retrieve answers. However, nRQL allows for defined acyclic queries. For instance, let us consider the following query:

$$MarriedMother(x) \leftarrow Mother(x) \ \wedge \ hasSpouse(x, y) \tag{3.14}$$

If the following query is defined in the knowledge base

$$Mother(x) \leftarrow Woman(x) \ \wedge \ hasChild(x, y)$$

then the query (3.14) can be rewritten as:

$$MarriedMother(x) \leftarrow Woman(x) \wedge hasChild(x, y) \wedge hasSpouse(x, z)$$

In spite of the defined queries feature, which is essentially a macro mechanism, nRQL is a query language and not a KR formalism for modeling domains. As a consequence, it is not a language applicable to the problem presented in Section 2.1.

Chapter 4

Preliminary Research

As we already saw, our goal is to find a KR formalism that resolves the problems described in Chapter 2 and satisfies the outlined requirements. In Chapter 3, we examined a number of formalisms and argued that they are not suitable for the discussed problem. In the current chapter, we present a formalism, which is work-in-progress and aims to address the aforementioned problem. First, we explain the concept recognition and graph composition problems in more detail; we use two examples involving chemical molecules for this purpose. Subsequently, we present a high-level graphical notation, which can be used to represent structured objects in a graph-theoretical way. Later on, we present a logic-based language that represents the same structures with the help of FOL-style formulas; we also show how the requirements described in Section 4.1 are satisfied by the presented formalism. Finally, we discuss what purposes the graphical notation and the logic-based language serve and what is the connection between them.

4.1 Two Essential Requirements

As we mentioned at the end of Chapter 2, one indispensable feature of the formalism we are aiming at is the ability to recognise concepts when their structure is encoded in the knowledge base; another necessary feature is the possibility to compose graphs, which represent objects with arbitrary complex structure. In the present section, we explain these two features using an example taken from the chemistry domain. In specific, we use molecules, which have the form of graphs because atoms correspond to vertices and bonds between atoms to edges.

4.1.1 Concept recognition

Figure 4.1 depicts the molecular structure of methane: a carbon atom connected through four bonds to four hydrogen atoms. As this is a typical ex-

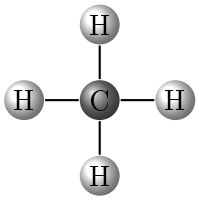


Figure 4.1: Molecule of methane.

ample of the kind of information we want to represent; the formalism should be able to encode the structure of Figure 4.1. Assume that the knowledge base contains this encoding; assume also that the knowledge base contains that there exists a molecule m with four hydrogen atoms, one carbon atom and four bonds between the carbon atom and the hydrogen atoms. The formalism should be designed in such a way that the knowledge base entails the fact that m is methane. In other words, we want methane to be *recognised*, when all the concept and role assertions that describe the methane structure are in the knowledge base. Furthermore, we require the methane concept to be recognised, only when the assertions correspond to methane and not when they correspond to a molecule with additional structure. E.g. if it is added to the knowledge base that the molecule m has one more hydrogen atom connected through a bond with the carbon atom (five hydrogen atoms in total), then the fact that m is methane should no longer be entailed.

4.1.2 Graph composition

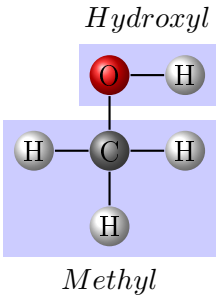


Figure 4.2: Molecule of methanole.

Figure 4.2 shows the inner structure of the methanole molecule: it consists of methyl (one carbon with three hydrogen atoms) and hydroxyl (oxygen with hydrogen) connected with a bond between the carbon and the oxygen atom. In a suitable formalism, we should be able to define a graph structure that encodes methanole not by referring to its atoms (hydrogen, carbon, oxygen) but by referring to its compounds (methyl, hydroxyl) which are already

encoded by a graph structure. That is, we need our language to permit us to formally define *composition* of graphs. Several complications should be taken into account in the definition of an appropriate formalism, e.g. a graph may contain several instances of the same graph, self-reference should be avoided etc.

4.2 Graph Notation

We now present a notation which can be used to encode objects with graph structure. The objects are described with the help of special graphs, which are inductively defined graph structures. Special graphs can either be simply graphs that consist of nodes and edges or graphs that use other special graphs as its structural components. Given that one graph might use several instances of the same graph as its components, we assign a different name to each graph instance so that we can distinguish between them. In the special graph definition, the set GI is the set of the graph instances names. The mapping μ from the set GI to the set of special graphs, is used to indicate for each graph instance, the special graph which it is an instance of. We call the notation *graph notation*; the purpose of graph notation is a language aiming to represent graph structures without the need of logical formulas, which are usually lengthy and illegible.

4.2.1 Definition

Let N_U and N_B be two countable infinite sets of unary and binary predicates. The set of *special graphs* is the smallest set, such that:

- Induction base: $(V, E, \emptyset, \emptyset, \lambda)$ is in the set of special graphs, where V is a finite set of natural numbers that denote vertices, E is a set of edges, such that $E \subseteq V \times V$ and λ is a labeling function, such that $\lambda : V \to \mathcal{P}(N_U \setminus \emptyset)$ and $\lambda : E \to \mathcal{P}(N_B \setminus \emptyset)$.
- Induction step: If $G_i = (V_i, E_i, GI_i, \mu_i, \lambda)$, where $1 \leq i \leq n$, are special graphs, N_{GI} is a countably infinite set of graph instance names, $GI \subseteq N_{GI}$ and $\mu : GI \to \{G_1, \ldots, G_n\}$, then G is in the set of special graphs, where $G = (V_N \cup V_{GI}, E_N \cup E_{GI}, GI, \mu, \lambda)$ and:
  - V_N is a finite set of natural numbers
  - $V_{GI} = \bigcup_{\hat{G} \in GI} \{(\hat{G}, v) \mid \mu(\hat{G}) = (V', E', GI', \mu', \lambda') \wedge v \in V'\}$
  - $E_N \subseteq (V_N \cup V_{GI}) \times (V_N \cup V_{GI})$

–

$$E_{GI} = \bigcup_{\hat{G} \in GI} \{((\hat{G}, v_1), (\hat{G}, v_2)) \mid \mu(\hat{G}) = (V', E', GI', \mu', \lambda') \wedge (v_1, v_2) \in E'\}$$

– $\lambda : V_N \cup V_{GI} \rightarrow \mathcal{P}(N_U \setminus \emptyset)$ and $\lambda : E_N \cup E_{GI} \rightarrow \mathcal{P}(N_B \setminus \emptyset)$.

4.2.2 Examples

Next, we use the graph notation to represent the molecules of methane and methanole. The methane molecule simply consists of atoms and bonds, whereas the methanole molecule comprises one methyl compound and one hydroxyl compound connected with a bond between them.

- The methane molecule is depicted in Figure 4.1:
  $G = (V, E, GI, \mu, \lambda)$
  $V = \{1, 2, 3, 4, 5\}$
  $E = \bigcup_{2 \leq i \leq 5} \{(1, i), (i, 1)\}$
  $GI = \emptyset$
  $\mu = \emptyset$
  $\lambda(1) = Carbon$
  $\lambda(i) = Hydrogen,\ 2 \leq i \leq 5$
  $\lambda((1, i)) = \lambda((i, 1)) = Bond,\ 2 \leq i \leq 5$
- The methanole molecule is depicted in Figure 4.2. First, we represent the methyl and hydroxyl as special graphs and, then, the methanole molecule as a special graph that uses a graph instance of methyl and a graph instance of hydroxyl for its definition.
  - Methyl
  - $G_{myl} = (V_{myl}, E_{myl}, \emptyset, \emptyset, \lambda_{myl})$
    $V_{myl} = \{1, 2, 3, 4\}$
    $E_{myl} = \bigcup_{2 \leq i \leq 4} \{(1, i), (i, 1)\}$
    $\lambda_{myl}(1) = Carbon$
    $\lambda_{myl}(i) = Hydrogen,\ 2 \leq i \leq 4$
    $\lambda_{myl}((1, i)) = \lambda_{myl}((i, 1)) = Bond,\ 2 \leq i \leq 4$
  - Hydroxyl
    $G_{hxl} = (V_{hxl}, E_{hxl}, \emptyset, \emptyset, \lambda_{hxl})$
    $V_{hxl} = \{1, 2\}$
    $E_{hxl} = \{(1, 2), (2, 1)\}$

$\lambda_{hxl}(1) = Oxygen$
$\lambda_{hxl}(2) = Hydrogen$
$\lambda_{hxl}((1,2)) = \lambda_{hxl}((2,1)) = Bond$

– Methanole
$G = (V, E, GI, \mu, \lambda)$
$V = \bigcup_{1 \leq i \leq 4, j=1,2} \{(GI_{myl}, i), (GI_{hxl}, j)\}$
$E = \bigcup_{2 \leq i \leq 4} \{((GI_{myl}, 1), (GI_{myl}, i)), ((GI_{myl}, i), (GI_{myl}, 1))\} \cup$
$\{((GI_{hxl}, 1), (GI_{hxl}, 2)), ((GI_{hxl}, 2), (GI_{hxl}, 1))\} \cup$
$\{((GI_{hxl}, 1), (GI_{myl}, 1)), ((GI_{myl}, 1), (GI_{hxl}, 1))\}$
$GI = \{GI_{myl}, GI_{hxl}\}$
$\mu(GI_{myl}) = G_{myl},\ \mu(GI_{hxl}) = G_{hxl}$
$\lambda((GI_{myl}, 1)) = Carbon$
$\lambda((GI_{myl}, 2)) = \lambda((GI_{myl}, 3)) = \lambda((GI_{myl}, 4)) = \lambda((GI_{hxl}, 2)) =$
$Hydrogen$
$\lambda((GI_{hxl}, 1)) = Oxygen$
$\lambda(((GI_{hxl}, 1), (GI_{hxl}, 2))) = \lambda(((GI_{hxl}, 2), (GI_{hxl}, 1))) =$
$\lambda(((GI_{myl}, 1), (GI_{myl}, j))) = \lambda(((GI_{myl}, j), (GI_{myl}, 1))) =$
$\lambda(((GI_{myl}, 1), (GI_{hxl}, 1))) = \lambda(((GI_{hxl}, 1), (GI_{myl}, 1)))$
$= Bond,\ 2 \leq j \leq 4$

4.3 Graph Logic

We now present a language that allows us to encode graph structures with the help of logical formulas. We call this language *graph logic*; its purpose is to describe the same graph structures that graph notation is representing but in a more low-level way, which is based on FOL.

The language has a FOL with equality syntax style, but it is function-free and only unary and binary predicates are allowed. In order to represent a labeled graph using logical formulas, we need to encode both the membership of nodes and edges to the graph, as well the labels that the nodes and edges bear. We can use a binary predicate to express membership of nodes to a graph (e.g. $hasNode(graph, c)$, $hasNode(graph, h)$) and unary (binary) predicates for the labels of nodes (edges) (e.g. $Carbon(c)$, $Hydrogen(h)$, $Bond(c, h)$). Nevertheless, it is not clear how we can express membership of an edge to a graph; we could use an expression of the form $hasEdge(graph, c, h)$ but our logic is limited to unary and binary predicates. In order to overcome this difficulty, we decide to consider a graph as a set of assertions about its nodes and edges and use a special predicate that indicates when a graph contains an assertion. We call this predicate GC (from graph contains); the expression $GC(g, a)$, where g is a term and a is an assertion intuitively means that g is a graph which contains the assertion a.

An assertion is a unary (binary) predicate atom about a node (edge) of the graph. GC encodes both the membership of a node (edge) to a graph, as well the label of the node (edge). With the help of GC, the graph mentioned earlier in the paragraph can be described by specifying the three assertions that the graph contains, i.e. $GC(graph, Carbon(c))$, $GC(graph, Hydrogen(h))$ and $GC(graph, Bond(c, h))$.

Another expressive feature that our logic should be equipped with is the ability to quantify over predicates and assertions; for instance we would like to quantify over the assertions that the graph contains or check whether a node belongs to a graph (e.g. equations (4.1) and (4.8) of Section 4.3.3). Nevertheless, we prefer to avoid second-order semantic structures in which variables range over domains of predicates and assertions constructed out of the domain of individuals; as it is explained by Chen et al. [8] second-order semantics is much more difficult to handle. Instead, we want the variables to range over names of predicates and names of assertions. To resolve this issue, we adopt HiLog-style semantics and instead of having a single domain of interpretation Δ^I, we distinguish three different sorts ${\Delta_C}^I$, ${\Delta_P}^I$, ${\Delta_A}^I$, the elements of which interpret terms, predicates and assertions, respectively. Additionally, we do not evaluate assertions by checking membership of elements to predicate interpretations; instead, we define mappings of assertions to members of ${\Delta_A}^I$ and, subsequently, define a subset of ${\Delta_A}^I$ as the set of "true" objects. A benefit of having a multi-sorted domain is that when we quantify e.g. over predicates we only consider members of ${\Delta_P}^I$ and not elements of the domain that intepret terms or assertions. Moreover, when the special predicate GC is interpreted, we avoid considering terms containing terms or assertions containing assertions, but we only consider terms (graphs) containing assertions, which is the only sensible combination that should be considered for the GC predicate.

4.3.1 Syntax

A *Graph-logic signature* is a tuple $L = (N_U, N_B, N_C, \{GC\})$, where N_U, N_B, N_C and $\{GC\}$ are countable and pairwise disjoint sets. We use the abbreviation $S(L) = N_U \cup N_B \cup N_C \cup \{GC\}$. N_U and N_B are the sets of *unary* and *binary predicates* respectively, N_C is the set of *constants* and $\{GC\}$ is a singleton set, where GC is the *graph containment predicate*. Let also N_V be a nonempty set of variables disjoint with $S(L)$ such that $N_V = N_{V_C} \cup N_{V_A} \cup N_{V_P}$ and $N_{V_C} \cap N_{V_A} = N_{V_C} \cap N_{V_P} = N_{V_A} \cap N_{V_P} = \emptyset$.

A *term of* L (or simply a *term*) is either a variable v ($v \in V_{N_C}$) or a constant c ($c \in V_C$). An *assertion of* L (or simply an *assertion*) is either a variable v ($v \in V_{N_A}$) or an expression of the form $U(t)$, $B(t, t')$, $P(t)$ or $P(t, t')$ where $U \in N_U$, $B \in N_B$, t, t' are terms and $P \in V_{N_P}$. An *atomic formula of* L (or simply an *atomic formula*) is either an assertion or an expression of the form $x \approx y$, where x and y are terms or assertions.

The set of *formulas of* L is the smallest set meeting the following conditions:

1. Any atomic formula of L is a formula of L.
2. If ϕ is a formula of L so is $\neg\phi$.
3. If ϕ and ψ are formulas L so are $\phi \wedge \psi$.
4. If t is a term of L and A is an assertion of L then $GC(t, A)$ is a formula of L.
5. If ϕ is a formula of L and $x \in N_{V_C}$, then $(\forall_C x : \phi)$ and $(\exists_C x : \phi)$ are formulas of L.
6. If ϕ is a formula of L and $x \in N_{V_A}$, then $(\forall_A x : \phi)$ and $(\exists_A x : \phi)$ are formulas of L.
7. If ϕ is a formula of L and $x \in N_{V_P}$, then $(\forall_P x : \phi)$ and $(\exists_P x : \phi)$ are formulas of L.

Lastly, a *theory* T *of* L is a set of formulas of L. We use the connectives $\vee$, $\not\approx$, $\leftarrow$, $\rightarrow$ and $\leftrightarrow$ as an abbreviation. We also omit the indices of the existential and universal quantifiers, when it is straightforward from the context whether they quantify over constants, predicates or assertions. Additionally, we abbreviate $\forall x_1 : \forall x_2 : \ldots \forall x_n$ to $\forall x_1, x_2, \ldots, x_n :$ (same for $\exists$) and we write $t_1 \not\approx t_2 \not\approx \ldots \not\approx t_n$ when the terms of a set $\{t_i\}_{1 \leq i \leq n}$ are mutually unequal to each other.

4.3.2 Semantics

An interpretation for a Graph Logic signature $L = (N_U, N_B, N_C, \{GC\})$ is a tuple $I = (\Delta^I, \cdot^I, {f_U}^I, {f_B}^I, GC^I, TRUE^I)$, where $\Delta^I = {\Delta_C}^I \cup {\Delta_P}^I \cup {\Delta_A}^I$ and ${\Delta_C}^I \cap {\Delta_P}^I = {\Delta_P}^I \cap {\Delta_A}^I = {\Delta_C}^I \cap {\Delta_A}^I = \emptyset$. ${\Delta_C}^I$, ${\Delta_P}^I$, ${\Delta_A}^I$ are non-empty sets and are called *domain* of *constants*, *predicates* and *assertions* respectively. The *interpretation function* assigns an element $c^I \in {\Delta_C}^I$ to each constant $c \in N_C$ and an element $P^I \in {\Delta_P}^I$ to each (unary or binary) predicate $P \in N_U \cup N_B$. The functions ${f_U}^I : {\Delta_P}^I \times {\Delta_C}^I \rightarrow {\Delta_A}^I$ and ${f_B}^I : {\Delta_P}^I \times {\Delta_C}^I \times {\Delta_C}^I \rightarrow {\Delta_A}^I$ return an element $a \in {\Delta_A}^I$ for each assertion of the form $U(c)$ or $B(c, c')$ respectively, where U^I, $B^I \in {\Delta_P}^I$ and c^I, $c'^I \in {\Delta_C}^I$. We also have that $GC^I \subseteq {\Delta_C}^I \times {\Delta_A}^I$ and $TRUE^I \subseteq {\Delta_A}^I$. Let also $\mu_C : N_{V_C} \rightarrow {\Delta_C}^I$, $\mu_P : N_{V_P} \rightarrow {\Delta_P}^I$ and $\mu_A : N_{V_A} \rightarrow {\Delta_A}^I$ be the mappings that assign members of Δ^I to each variable. We denote with μ all the three mappings. If $v \in N_V$, we say that the mapping μ' is v-variant of the mapping μ iff μ and μ' assign the same elements of the domain to every variable except possibly v.

Let t be a term. We define:

Table 4.1: Satisfaction of Formulas for Graph Logic

$I, \mu \models U(x)$	iff	$U(x)^{\mu,I} \in TRUE^I$
$I, \mu \models B(x, y)$	iff	$B(x, y)^{\mu,I} \in TRUE^I$
$I, \mu \models x \approx y$	iff	$x^{\mu,I} = y^{\mu,I}$
$I, \mu \models GC(t, A)$	iff	$(t^{\mu,I}, A^{\mu,I}) \in GC^I$
$I, \mu \models \forall_C x : \phi$	iff	$I, \mu' \models \phi$ for every mapping μ' which is an x-variant of μ
$I, \mu \models \exists_C x : \phi$	iff	$I, \mu' \models \phi$ for some mapping μ' which is an x-variant of μ
$I, \mu \models \forall_P x : \phi$	iff	$I, \mu' \models \phi$ for every mapping μ' which is an x-variant of μ
$I, \mu \models \exists_P x : \phi$	iff	$I, \mu' \models \phi$ for some mapping μ' which is an x-variant of μ
$I, \mu \models \forall_A x : \phi$	iff	$I, \mu' \models \phi$ for every mapping μ' which is an x-variant of μ
$I, \mu \models \exists_A x : \phi$	iff	$I, \mu' \models \phi$ for some mapping μ' which is an x-variant of μ
$I, \mu \models \neg\phi$	iff	$I, \mu \not\models \phi$
$I, \mu \models \phi \wedge \psi$	iff	$I, \mu \models \phi$ and $I, \mu \models \psi$
$I \models \phi$	iff	$I, \mu \models \phi$ for all mappings μ
$I \models T$	iff	$I \models \phi$ for all $\phi \in T$
$T \models \phi$	iff	for all interpretations I, if $I \models T$, then $I \models \phi$

- If $t \in N_C$, then $t^{\mu,I} = t^I$.
- If $t \in N_{V_C}$, then $t^{\mu,I} = \mu_C(t)$.

Let A be an assertion. We define:

- If $A \in N_{V_A}$, then $A^{\mu,I} = \mu_A(A)$.
- If $A = U(x)$, $U \in N_U$ and x is a term, then $U(x)^{\mu,I} = {f_U}^I(U^I, x^{\mu,I})$.
- If $A = B(x, y)$, $B \in N_B$ and x, y are terms, then we have that $B(x, y)^{\mu,I} = {f_B}^I(B^I, x^{\mu,I}, y^{\mu,I})$.
- If $A = P(x)$, $P \in N_{V_P}$ and x is a term, $P(x)^{\mu,I} = {f_U}^I(\mu_P(P), x^{\mu,I})$.
- If $A = P(x, y)$, $P \in N_{V_P}$ and x, y are terms, then we have that $P(x, y)^{\mu,I} = {f_B}^I(\mu_P(P), x^{\mu,I}, y^{\mu,I})$.

We define satisfaction of formulas in an interpretation (and a mapping) in Table 4.1.

4.3.3 Examples

Subsequently, we use the graph logic to represent the methane and methanole molecules, which where previously described with our graph notation.

Molecules with atoms and bonds

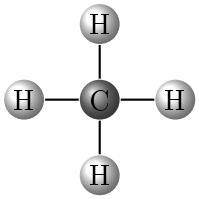


Figure 4.3: Molecule of methane.

Figure 4.3 depicts the molecule of methane: a carbon atom is connected through four single bonds to four hydrogen atoms. Formula (4.1) describes the methane structure as a graph which contains five atom assertions, four bond assertions and for which the constants which serve as atoms are pairwise unequal. Formula (4.2) says that *SingleBond* is a symmetric binary predicate.

$$\begin{aligned} \forall x : Methane(x) \leftrightarrow \exists c, h_1, h_2, h_3, h_4 : c \not\approx h_1 \not\approx h_2 \not\approx h_3 \not\approx h_4 \wedge \\ \forall w : GC(x,w) \leftrightarrow \bigvee_{i=1,\ldots,4} (w \approx Hydrogen(h_i)) \vee (w \approx Carbon(c)) \\ \vee \bigvee_{i=1,\ldots,4} (w = Bond(c,h_i)) \end{aligned} \tag{4.1}$$

$$\forall x, y : Bond(x,y) \leftrightarrow Bond(y,x) \tag{4.2}$$

Formula (4.1) addresses the concept recognition problem: if m contains the five atom and the four bond assertions the fact $Methane(m)$ is true. Note that if we add further structure to the methane molecule and consider the assertions that correspond to the structure (e.g. Figure 4.4) the fact $Methane(m)$ is no longer true. The reason is that the right-hand side of Formula (4.1) is no longer true. There exists an assertion a (e.g. $a \approx Hydrogen(h_5)$) such that $GC(x,a)$ is true but a does not match any of the right-hand side disjuncts ($Hydrogen(h_5) \not\approx Carbon(c)$ and for $1 \leq i \leq 4$, $Hydrogen(h_5) \not\approx Hydrogen(h_i)$ and $Hydrogen(h_5) \not\approx SingleBond(c,h_i)$).

Similarly, if some of the existing structure of the molecule is removed (e.g. Figure 4.5) the graph is not recognised as methane. Again, the double implication is false, because there exists an assertion (e.g. $Hydrogen(h_4)$,

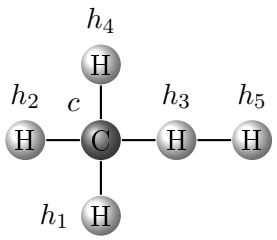


Figure 4.4: Methane molecule with more structure.

where $h_1 \not\approx h_2 \not\approx h_3 \not\approx h_4$) that satisfies the disjunction (since the disjunct $Hydrogen(h_4) \approx Hydrogen(h_4)$ is satisfied) but which is not contained by the graph ($GC(x, Hydrogen(h_4))$ is false).

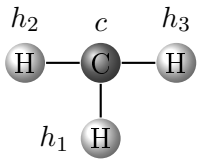


Figure 4.5: Methane molecule with less structure.

Molecules with inner structure

The methanole molecule is illustrated by Figure 4.6: it consists of a hydroxyl compound connected to a methyl compound through a bond which connects hydroxyl's oxygen with methyl's carbon. Formula (4.3) describes methanole as the disjoint union of two graphs (the graph that represents hydroxyl and the graph that represents methyl) plus the single bond that links hydroxyl's oxygen with methyl's carbon.

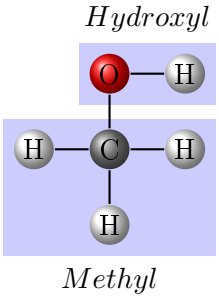


Figure 4.6: Molecule of methanole.

$$\begin{aligned}\forall x : Methanole(x) \leftrightarrow \exists h, m : h \not\approx m \wedge Hydroxyl(h) \wedge \\ Methyl(m) \wedge Disjoint(h, m) \wedge \\ \forall w : GC(x, w) \leftrightarrow GC(h, w) \vee GC(m, w) \vee \\ (\exists o, c : FreeOxygen(h, o) \wedge FreeCarbon(m, c) \\ \wedge o \not\approx c \wedge (w \approx SingleBond(o, c)))\end{aligned} \tag{4.3}$$

Hydroxyl and methyl are defined in the same way as methane was defined: a set of assertions about the nodes and the edges that the corresponding graph contains.

$$\begin{aligned}\forall x : Hydroxyl(x) \leftrightarrow \exists o, h : o \not\approx h \wedge \\ \forall w : GC(x, w) \leftrightarrow (w \approx Hydrogen(h)) \vee (w \approx Oxygen(o)) \\ \vee (w \approx SingleBond(h, o))\end{aligned} \tag{4.4}$$

$$\begin{aligned}\forall x : Methyl(x) \leftrightarrow \exists c, h_1, h_2, h_3 : c \not\approx h_1 \not\approx h_2 \not\approx h_3 \wedge \\ \forall w : GC(x, w) \leftrightarrow \bigvee_{i=1,\dots,3} (w \approx Hydrogen(h_i)) \vee (w \approx Carbon(c)) \\ \vee \bigvee_{i=1,\dots,3} (w \approx SingleBond(c, h_i))\end{aligned} \tag{4.5}$$

We now need to define the binary predicates $FreeOxygen$ and $FreeCarbon$ which retrieve the suitable node from the hydroxyl and methyl graph respectively. For this specific case, the definition of the predicates is straightforward because hydroxyl (methyl) has only one oxygen (carbon) atom.

$$\forall g, o : FreeOxygen(g, o) \leftrightarrow GC(g, Oxygen(o)) \wedge Hydroxyl(g) \tag{4.6}$$

$$\forall g, c : FreeCarbon(g, c) \leftrightarrow GC(g, Carbon(c)) \wedge Methyl(g) \tag{4.7}$$

Finally, we need to make sure that there is no common node between the two compounds, otherwise a structure with a common hydrogen atom (e.g. Figure 4.7) is falsely recognized as methanole. In order to do so, we need to define a new predicate, VIG (vertex in graph), which indicates when a vertex v is in a graph g.

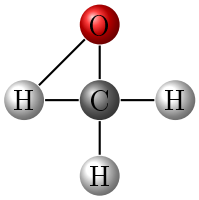


Figure 4.7: Molecule which is not methanole.

$$\begin{aligned}\forall x, v : VIG(v, g) \leftrightarrow & \exists U : GC(g, U(v)) \vee \\ & \exists B, v' : GC(g, B(v, v')) \vee GC(g, B(v', v))\end{aligned} \tag{4.8}$$

Given the VIG predicate, we define the disjointness of two graphs by requiring for all vertices not to be in both graphs at the same time.

$$\forall g_1, g_2 : Disjoint(g_1, g_2) \leftrightarrow \forall v(\neg VIG(v, g_1) \vee \neg VIG(v, g_2)) \tag{4.9}$$

Methanole could have been described in a similar way to methane. In particular, an alternative definition for methanole would be:

$$\begin{aligned}\forall x : Methanole'(x) \leftrightarrow \exists c, o, h_1, h_2, h_3, h_4 : c \not\approx o \not\approx h_1 \not\approx h_2 \not\approx h_3 \not\approx h_4 \wedge \\ \forall w : GC(x, w) \leftrightarrow (w \approx Oxygen(o)) \vee (w \approx Carbon(c)) \vee \\ \bigvee_{i=1,\ldots,4} (w \approx Hydrogen(h_i)) \vee \\ \bigvee_{i=1,\ldots,3} (w \approx SingleBond(c, h_i)) \vee \\ (w \approx SingleBond(c, o)) \vee \\ (w \approx SingleBond(o, h_4))\end{aligned} \tag{4.10}$$

If we compare the two approaches, there is a clear advantage in dealing with compounds (e.g. hydroxyl, methyl) instead of atoms (e.g. hydrogen, carbon): it is easier to describe molecules with tens or even hundreds of atoms by using a hierarchical structure rather than enumerating every single atom and bond. Nevertheless, we need to make sure that formulas (4.3) and (4.10) describe the same graph structure. In particular, we need to show that:

$$\forall x : Methanole(x) \leftrightarrow Methanole'(x)$$

In order to prove that, we need to add axioms such that for every possible subset of assertions (apart from the empty set) there exists a graph which contains them. This is the only way to ensure that there are graphs which contain those sets of assertions that are necessary to form a compound such as hydroxyl (from the assertions $Hydrogen(h)$, $Oxygen(o)$ and $SingleBond(h, o)$) or methyl (from the assertions $Carbon(c)$, $Hydrogen(h_i)$ and $SingleBond(c, h_i)$ for $1 \leq i \leq 3$). We enforce this requirement by introducing formulas (4.11) and (4.12):

$$\forall w_{single} : \exists g_{single} : \forall w : GC(g_{single}, w) \leftrightarrow w_{single} \approx w \tag{4.11}$$

$$\forall g_{init}, w_{add} : \exists g_{add} : \forall w : GC(g_{add}, w) \leftrightarrow (w_{add} \approx w) \vee GC(g_{init}, w) \tag{4.12}$$

Formula (4.11) makes sure that for every assertion w_{single} there exists a graph which contains that and only that assertion. Formula (4.12), on the other hand, makes sure that for every graph g_{init} and for every assertion w_{add} which is not contained by g_{init} there exists a new graph g_{add} which contains the assertions of g_{init} plus the assertion w_{add} and nothing else than that. The two formulas axiomatise the fact that for every set of assertions (excluding the empty set) there exists a graph which contains those and only those assertions.

After defining (4.11) and (4.12), the logical equivalence of the two different methanole representations can be proved by showing (4.13), where T includes (4.2)-(4.12):

$$T \models \forall x : Methanole(x) \leftrightarrow Methanole'(x) \tag{4.13}$$

Proof. (Sketch) (4.13) can be proved by showing that for every interpretation I that satisfies T, if $f_1^I(Methanole^I, m^I) \in TRUE^I$, then we will have $f_1^I(Methanole'^I, m^I) \in TRUE^I$ (and the inverse). We prove that by considering a model I, such that, for $m^I \in {\Delta_C}^I$, $f_1^I(Methanole^I, m^I) \in TRUE^I$. Since I satisfies axiom (4.3) and $f_1^I(Methanole^I, m^I) \in TRUE^I$, I also satisfies the right hand side of the axiom (4.3), when x is interpreted by m^I. From this and the axioms that T contains, we can prove that I satisfies the right-hand side of the axiom (4.10), when x is interpreted by m^I, and, thus, that $f_1^I(Methanole'^I, m^I) \in TRUE^I$. The inverse is proved in a similar way. □

Having proved that, we have shown that we can use graph logic to define complex objects (e.g. (4.5) and (4.4) for methyl and hyrdoxyl respectively) and then reuse these definitions in order to define more complex objects(e.g. (4.3) uses methyl and hydroxyl which are previously defined). Therefore, graph logic permits us to describe graph composition with the help of logic-based formulas.

4.4 Linking Graph Notation and Graph Logic

As we observe, graph notation and graph logic are two different languages that represent the same structures. The reason for introducing these two distinct languages is that they serve different purposes.

The objective of graph notation is to provide a high-level formal language, such that it is relatively straightforward to map a graph structure to graph notation not only for computer scientists but also for scientists of other areas, such as chemists. Graph notation encodes graph structures without using any burdensome first-order logic formulas and, it is an easier language to use than graph logic.

Graph logic, on the other hand, represents graphs with the help of first-order logic. Given that one of our major goals is to provide a proof procedure for the language, it is clearly more feasible to develop an inference algorithm for a logic-based language, rather than for the graph notation. Moreover, a FOL-like formalism, such as graph logic, is easier to be combined with other logic-based languages, such as description logics, and allow for hybrid knowledge bases.

In order to make graph notation and graph logic useful in practice, we need to define a way that appropriately translates graph notation to graph logic. The translation process is not presented in this report, because it has not been fully specified yet and it is ongoing work.

Chapter 5

Completed and Future Work

In the last chapter, we conclude our analysis by providing an account of the undertaken and future research. We discuss what are the goals that we plan to achieve in the future and we also suggest a time schedule that spans the first year and the remaining time of the DPhil.

5.1 Research So Far

The tasks that have been completed so far include the following:

- We have specified the problem that we are going to deal with and identified the most important requirements that a suggested solution should satisfy.
- We have explored various domains of knowledge and come across use cases which are analogous to the problem we are interested in. Therefore, the problem is a frequently encountered and significant one.
- We have conducted an extensive literature review of KR formalisms that might be used to solve the examined problem and analysed why they are not suitable for our setting.
- We have designed the syntax and semantics of a logic tailored to our problem and its requirements. Additionally, we have shown how the problems of graph recognition and composition are resolved with the use of our logic.
- We have defined a graphical notation for the representation of graph structures that allows for graph composition.

5.2 Future Plan

In the future, we will pursuit the following goals:

- Specify formally how the graph notation can be translated to formulas of graph logic.
- Design an efficient proof procedure for our logic, which is sound, complete and terminating.
- Develop a system that implements the reasoning algorithm.
- Optimise and evaluate the system by choosing and applying appropriate benchmarking methods.

We now describe an additional problem that, if there is adequate time, we would like to deal with in the future. One of the features of graph logic is that the encoded structures need to be of fixed size. The formalisms presented in Chapter 4 aim to represent simple graphs or graphs which are composed from instances of other graphs, but in both cases the size of the graph is specified upfront. In other words, neither graph logic nor graph notation are suitable for graphs that include in their structure one or more compounds such that the number of compounds is not known in advance. We give an example of such a structure, taken from the chemistry domain. Figure 5.1 depicts the molecule of monohydric alcohol, which is an arbitrarily long chemical molecule: one or more $H - C - H$ blocks are connected to each other through a bond between the carbons and form a chain, while a hydroxyl and a hydrogen are attached to the two extremes of the chain.

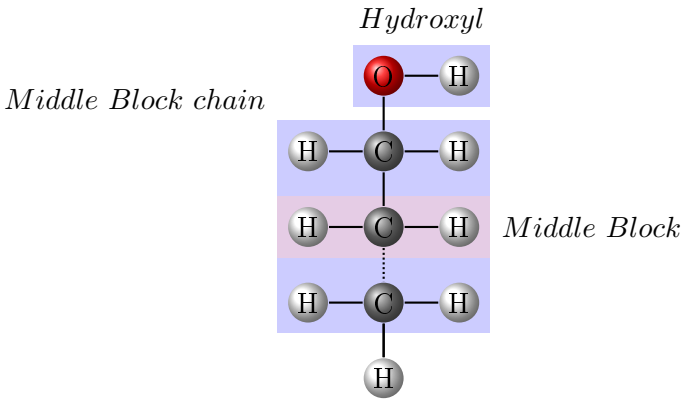


Figure 5.1: Molecule of monohydric alcohol.

With graph logic in its current form it is not possible to represent molecules, such as monohydric alcohol, because the number of $H - C - H$ parts in monohydric alcohol varies and is not known beforehand. However, it can be investigated whether the language can be extended in such a way that unboundedly large (but finite) structures can be expressed; the major difficulty of this problem is that an arithmetic induction definition is needed and it is

not possible to axiomatise arithmetic induction in first-order logic. Instead of that, an axiom schema is required that contains a separate axiom for each possible predicate or, alternatively, a second-order formula that includes a quantification over predicates.

In the future, we will try to modify our graph logic language in order to represent unbounded size graph structures. If we successfully tackle the problem, we shall appropriately extend the proof procedure, implementation and evaluation to also cover this case.

5.3 Time schedule

The following time schedule briefly describes the fulfilled and planned tasks from the starting date until the end of the DPhil. Some of them are not directly related with the material discussed in this report but give an account of what the first year was spent on.

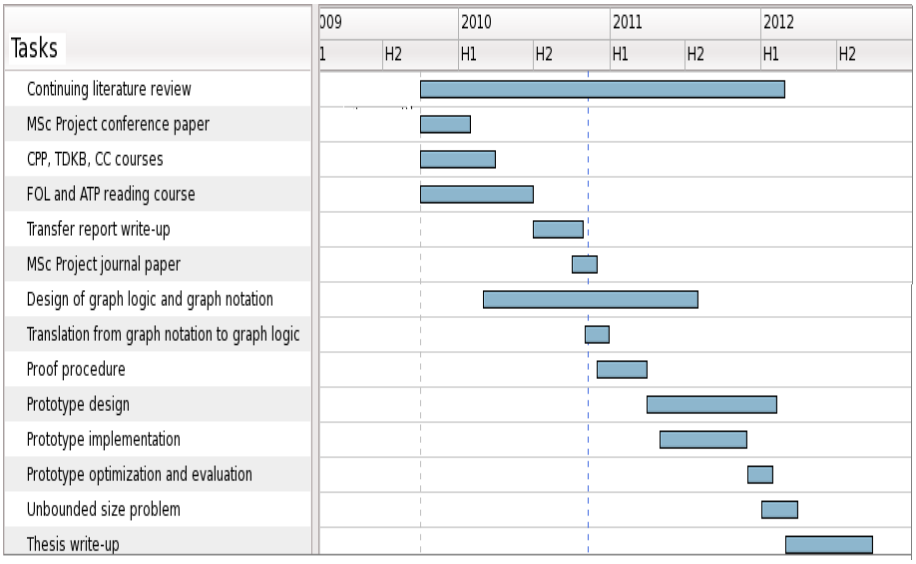


Figure 5.2: Gantt chart of past and future tasks

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Awny Alnusair and Tian Zhao. Using Ontology Reasoning for Reverse Engineering Design Patterns. In Sudipto Ghosh, editor, *MoDELS Workshops*, volume 6002 of *Lecture Notes in Computer Science*, pages 344–358. Springer, 2009.

[3] Grigoris Antoniou. Non-monotonic reasoning, MIT Press, 1997, ISBN 0-262-01157-3.

[4] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2007.

[5] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.

[6] Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation. *J. Log. Program.*, 19/20:73–148, 1994.

[7] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Eql-lite: Effective first-order query processing in description logics. In *In Proc. of IJCAI 2007*, pages 274–279.

[8] Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A Foundation for Higher-Order Logic Programming. *J. Log. Program.*, 15(3):187–230, 1993.

[9] Kirill Degtyarenko, Paula de Matos, Marcus Ennis, Janna Hastings, Martin Zbinden, Alan McNaught, Rafael Alcántara, Michael Darsow, Mickaël Guedj, and Michael Ashburner. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research*, 36(Database-Issue):344–350, 2008.

[10] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[11] Paolo Ferraris and Vladimir Lifschitz. Mathematical Foundations of Answer Set Programming. In Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! (1)*, pages 615–664. College Publications, 2005.

[12] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[13] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[14] Michael Gelfond. Answer Sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 285–310. Elsevier Science, 2007.

[15] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics For Logic Programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP)*, pages 1070–1080. MIT Press, 1988.

[16] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical report, Stanford University, Stanford, CA, USA, 1992.

[17] Racer Systems GmbH and Co. KG. RacerPro UserâĂŹs Guide, December 2005.

[18] Antoon Goderis, Ulrike Sattler, and Carole Goble. Applying descriptions logics for workflow reuse and repurposing. In *International Description Logics Workshop*, Edinburgh, Scotland, 2005.

[19] Henson Graves. Representing Product Designs Using a Description Graph Extension to OWL 2. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[20] Henson Graves and Ian Horrocks. Application of OWL 1.1 to Systems Engineering.

[21] Volker Haarslev, Ralf MÃűller, and Michael Wessel. Querying the Semantic Web with Racer + nRQL. In *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADLâĂŹ04*, 2004.

[22] Janna Hastings, Michel Dumontier, Duncan Hull1, Matthew Horridge, Christoph Steinbeck, Ulrike Sattler, Robert Stevens, Tertia Hörne, and Katarina Britz. Representing Chemicals using OWL, Description Graphs and Rules. In *OWLED*, CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[23] Rinke Hoekstra. Use of OWL in the Legal Domain (Statement of Interest). In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of OWL: Experiences and Directions (OWLED 2008 DC)*, Washington, DC (metro), April 2008.

[24] Rinke Hoekstra. *Ontology Representation - Design Patterns and Ontologies that Make Sense*, volume 197. IOS Press, 2009.

[25] Rinke Hoekstra and Joost Breuker. Polishing Diamonds in OWL2. In Aldo Gangemi and Jérôme Euzenat, editors, *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)*, LNAI/LNCS. Springer Verlag, October 2008.

[26] Duncan Hull. GO faster ChEBI with Reasonable Biochemistry. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[27] International Standards Organisation. ISO/IEC (2007) 'ISO/IEC 24707:2007 - Information technology - Common Logic (CL) - A framework for a family of logic-based languages', 2007. Geneva,Switzerland.

[28] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42:741–843, 1995.

[29] Michael Kifer and Georg Lausen. F-Logic: A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *SIGMOD Conference*, pages 134–146. ACM Press, 1989.

[30] Mykola Konyk, Alexander De Leon Battista, and Michel Dumontier. Chemical Knowledge for the Semantic Web. In Amos Bairoch, Sarah Cohen Boulakia, and Christine Froidevaux, editors, *DILS*, volume 5109 of *Lecture Notes in Computer Science*, pages 169–176. Springer, 2008.

[31] Vladimir Lifschitz. What Is Answer Set Programming? In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 1594–1597. AAAI Press, 2008.

[32] Wiktor Marek and Miroslaw Truszczynski. Autoepistemic logic. *J. ACM*, 38:587–618, July 1991.

[33] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulirke Sattler. Representing Structured Objects using Description Graphs. In Gerhard Brewka and Jérôme Lang, editors, *Proc. of the 11th Int. Joint Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 296–306, Sydney, NSW, Australia, August 16–19 2008. AAAI Press.

[34] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Modeling Ontologies Using OWL, Description Graphs, and Rules. In Alan Ruttenberg, Ulrile Sattler, and Cathy Dolbear, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany, October 26–27 2008.

[35] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Representing Ontologies Using Description Logics, Description Graphs, and Rules. *Artificial Intelligence*, 173(14):1275–1309, 2009.

[36] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured Objects in OWL: Representation and Reasoning. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proc. of the 17th Int. World Wide Web Conference (WWW 2008)*, pages 555–564, Beijing, China, April 21–25 2008. ACM Press.

[37] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. The Representation of Structured Objects in DLs using Description Graphs. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proc. of the 21st Int. Workshop on Description Logics (DL 2008)*, volume 353 of *CEUR Workshop Proceedings*, Dresden, Germany, May 13–16 2008.

[38] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. The Representation of Structured Objects in DLs using Description Graphs. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proc. of the 21st Int. Workshop on Description Logics (DL 2008)*, volume 353 of *CEUR Workshop Proceedings*, Dresden, Germany, May 13–16 2008.

[39] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *Proc. of the 5th Int. Semantic Web Conference (ISWC 2006)*, volume 4273 of *LNCS*, pages 501–514, Athens, GA, USA, November 5–9 2006. Springer.

[40] Boris Motik and Riccardo Rosati. A Faithful Integration of Description Logics with Logic Programming. In Manuela M. Veloso, editor, *Proc. of the 20th Int. Joint Conference on Artificial Intelligence (IJCAI*

2007), pages 477–482, Hyderabad, India, January 6–12 2007. Morgan Kaufmann Publishers.

[41] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.

[42] Ontoprise. How to write F-Logic Programs, September 2008.

[43] Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[44] Kavitha Srinivas. OWL Reasoning in the Real World: Searching for Godot. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[45] Holger Stenzhorn, Kavitha Srinivas, Matthias Samwald, and Alan Ruttenberg. Simplifying access to large-scale health care and life sciences datasets. In *ESWC'08: Proceedings of the 5th European semantic web conference on The semantic web*, pages 864–868, Berlin, Heidelberg, 2008. Springer-Verlag.

[46] Saskia van de Ven, Rinke Hoekstra, Joost Breuker, Lars Wortel, and Abdallah El-Ali. Judging Amy: Automated Legal Assessment using OWL 2. In *Proceedings of OWL: Experiences and Directions (OWLED 2008 EU)*, October 2008.

[47] Moshe Y. Vardi. Why is modal logic so robustly decidable? In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1996.

[48] Natalia Villanueva-Rosales and Michel Dumontier. Describing Chemical Functional Groups in OWL-DL for the Classification of Chemical Compounds. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[49] Michael Wessel, Marko Luther, and Ralf Möller. What Happened to Bob? Semantic Data Mining of Context Histories. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[50] Michael Wessel and Ralf Möller. A High Performance Semantic Web Query Answering Engine. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Description Logics*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.