OWL 2 Profiles: An Introduction to Lightweight Ontology Languages

Markus Krötzsch

Department of Computer Science, University of Oxford, UK
markus.kroetzsch@cs.ox.ac.uk

Abstract. This chapter gives an extended introduction to the lightweight profiles OWL EL, OWL QL, and OWL RL of the Web Ontology Language OWL. The three ontology language standards are sublanguages of OWL DL that are restricted in ways that significantly simplify ontological reasoning. Compared to OWL DL as a whole, reasoning algorithms for the OWL profiles show higher performance, are easier to implement, and can scale to larger amounts of data. Since ontological reasoning is of great importance for designing and deploying OWL ontologies, the profiles are highly attractive for many applications. These advantages come at a price: various modelling features of OWL are not available in all or some of the OWL profiles. Moreover, the profiles are mutually incomparable in the sense that each of them offers a combination of features that is available in none of the others. This chapter provides an overview of these differences and explains why some of them are essential to retain the desired properties. To this end, we recall the relationship between OWL and description logics (DLs), and show how each of the profiles is typically treated in reasoning algorithms.

1 Introduction

The Web Ontology Language OWL has been standardised by the World Wide Web Consortium (W3C) as a powerful knowledge representation language for the Web. In spite of its name, OWL is not confined to the Web, and it has indeed been applied successfully for knowledge modelling in many application areas. Modelling information in OWL has two practical benefits: as a descriptive language, it can be used to express expert knowledge in a formal way, and as a logical language, it can be used to draw conclusions from this knowledge. The second aspect is what distinguishes OWL from other modelling languages such as UML. However, computing all interesting logical conclusions of an OWL ontology can be a challenging problem, and reasoning is typically multi-exponential or even undecidable.

To address this problem, the recent update *OWL 2* of the W3C standard introduced three *profiles*: OWL EL, OWL RL, and OWL QL. These lightweight sublanguages of OWL restrict the available modelling features in order to simplify reasoning. This has led to large improvements in performance and scalability, which has made the OWL 2 profiles very attractive for practitioners. OWL EL is used in huge biomedical ontologies, OWL RL became the preferred approach for reasoning with Web data, and OWL QL provides database applications with an ontological data access layer. The most important computation tasks are *tractable* in each of these cases, that is, they can be performed in polynomial time.

Besides their practical importance, however, there are many further reasons why the OWL profiles are of special interest to researchers and practitioners:

- More Understandable. The lower complexity of reasoning also leads to algorithms that are conceptually simpler than general reasoning techniques for OWL.
- Simpler Implementation. Simpler reasoning algorithms are easier to implement correctly and (with some further effort) also efficiently.
- Better Tool Support. Users of OWL profiles have more reasoners to choose from. Specialised implementations often provide best performance, but general-purpose OWL reasoners can also be used.
- Easier to Extend. Simpler ontology languages are easier to extend with new features, due to simpler algorithms and less complex semantic interactions.

In spite of the many motivations for studying the tractable profiles of OWL, the available resources for learning about this subject are very limited. Indeed, the interested reader is largely left with the study of the official W3C standardisation documents, which are comprehensive and accurate, but also laden with technical details. Alternatively, one can resort to a plethora of academic papers that cover many related topics individually. Often the first task of the reader is to understand how a given work relates to the OWL standard at all. In any case, information remains distributed and usually focussed on one of the OWL profiles. Indeed, even researchers often have only a rather partial understanding of how the three OWL profiles compare to each other. This leads to unnecessary misconceptions among academics and practitioners alike.

This text sets out to tackle this problem by providing a comprehensive first introduction to the OWL profiles. Our overall premise is that the advantages and limitations of any one profile can best be understood by comparing it to the other two. The result should be a fresh perspective that opens up new possibilities: instead of depending on the (sometimes coincidental) design choices made when defining the OWL profiles, one can freely combine modelling features to suit the need of one's very own application or research. The OWL specification allows us to do this without giving up standard conformance or interoperability. The key for enjoying this flexibility is to understand the constraints that are at the very heart of the current language design.

1.1 Goals of this Document

The purpose of this text is to give a comprehensive introduction to the three lightweight profiles of OWL 2, with a particular focus on expressivity and efficient reasoning. A central goal is to explain *how* the three profiles differ, and *why* these differences are important to retain the desired properties. Our preferred perspective will be that of a computer scientist or ontology engineer who wants to gain a deeper understanding of the characteristics of the three profiles. This will not stop us, however, from discussing formal properties whenever they are important for practical usage or implementation. In particular, we are interested in the correctness of algorithms and in the complexity of relevant computational problems.

The text does not intend to be a general introduction to OWL (it omits practical aspects such as URIs, RDF translation, syntactic forms, and advanced features), or to description logics (it omits many advanced topics and formal details). Moreover, we mainly focus on the conceptual basics of language design and reasoning. Much could be said about practical aspects of implementation and optimisation, but this is well beyond the scope of one lecture. Readers who want to deepen their understanding of these topics can find many pointers to further literature in Section 6.

Intended Audience This text addresses two main audiences: practitioners who want to gain a deeper understanding of the OWL profiles, and researchers or students who are interested in tractable knowledge representation and reasoning. Hence the text is largely self-contained and does not assume prior knowledge in formal logics or ontology languages, although some familiarity with these topics will come in handy. However, a graduate-level understanding of basic computer science topics is generally assumed. For the benefit of advanced readers, we usually include full details and formal arguments, which can easily be skipped on a first reading.

How to Read this Text The text largely follows a linear structure, but still allows sections to be skipped on a first reading. Sections are relatively short in general and are concluded by a short summary of their essential content. We include various proofs, mainly for their didactic value of explaining why *exactly* something is the case. They can easily be skipped by the hurried reader. *Examples* and *remarks* are presented in visually distinguished boxes; both can typically be skipped without loss of continuity.

1.2 Overview

The remaining sections are organised as follows:

Section 2: An Introduction to OWL In this section, we take a first look at the Web Ontology Language OWL in general. We introduce the most important reasoning tasks for OWL ontologies, and discuss the computational challenges that they create. A number of expressive features of OWL are introduced by means of examples in the OWL *Functional-Style Syntax*. The more concise *description logic* syntax is introduced as a convenient shorter notation for OWL axioms. Most of our discussions are based on a rather limited set of OWL features, that are encompassed by the description logic $\mathcal{ALCI}$. Finally, we discuss the two formal semantics of OWL – *Direct Semantics* and *RDF-Based Semantics* – and explain how they relate to each other.

Section 3: Reasoning in the OWL Profiles Here we introduce for the first time the OWL Profiles EL, RL, and QL by means of three very simple ontology languages $\mathcal{EL}\mathit{tiny}$, $\mathcal{RL}\mathit{tiny}$, and $\mathcal{QL}\mathit{tiny}$. We then explain how reasoning tasks are typically solved for each of these languages. We begin with $\mathcal{RL}\mathit{tiny}$, for which we present a saturation-based instance retrieval calculus that is based on a simple set of inference rules. For $\mathcal{EL}\mathit{tiny}$, we then develop a classification calculus that works in a similar fashion, although it computes subclass inclusion axioms rather than instance assertions. The task we consider for $\mathcal{QL}\mathit{tiny}$ is query answering, a generalised form of instance retrieval. Our reasoning method of choice in this case is query rewriting, which is an interesting alternative to the saturation-based approaches in EL and RL. In each of the three cases, we show that the proposed methods are correct, and a number of interesting proof techniques are introduced in the process.

Section 4: The Limits of Lightweight Ontology Languages We now ask how far the expressive power of the lightweight profiles of OWL can be extended without loosing their good properties (especially the possibility to use the polynomial time reasoning methods of Section 3). We will see that some extensions are indeed possible, and that even the official OWL standard is overly restrictive in various cases. However, many extensions lead to considerably higher computational complexities. To explain this, we first recall some basic ideas of complexity theory, and show that unrestricted class unions must make reasoning NP-hard. We then demonstrate that the combination of any two of the three OWL profiles leads to an even higher exponential reasoning complexity. This result also justifies the existence of three different profiles. To show this, we use an ontology to simulate the computation of an *Alternating Turing Machine*, which is an interesting technique in its own right.

Section 5: Advanced Modelling Features This short section completes the overview of the three OWL profiles by discussing a number of additional features that had not been introduced before. In most cases, these features do not significantly change the way in which reasoning is implemented, and we do not present reasoning methods for them.

Section 6: Summary and Further Reading We conclude with a short summary of the main insights, and provide many pointers to related literature. References are avoided in the rest of the text.

2 An Introduction to OWL

2.1 The Web Ontology Language

The Web Ontology Language OWL is a formal language for expressing ontologies. In computer science, an ontology is a description of knowledge about a domain of interest, the core of which is a machine-processable specification with a formally defined meaning.[1] This is best explained by an example.

Example 1. A classical kind of ontology are biological taxonomies as used to classify species. For example, the species of house cats (*Felis catus*) belongs to the class of mammals (*Mammalia*), i.e., every instance of the former is also an instance of the latter. In OWL, we could formally express this as follows:

SubClassOf(*FelisCatus Mammalia*)

This allows us to formally state a subclass relation in a way that is understood by OWL reasoners. Note that this formal relationship does not capture all the *knowledge* that we intend to express

[1] The term is derived from the philosophical discipline of *Ontology* – the study of existence and the basic relationships between the things that are – since a basic purpose of ontologies in computer science is to describe the existing entities and their interrelation.

here: it is also important to know what is actually *meant* by the classes *FelisCatus* and *Mammalia*. This aspect is essential if we want to view ontologies as representations of knowledge about the world, but it cannot be captured formally.

OWL statements as in the previous example are known as *OWL axioms*. There are many additional kinds of axioms that one can express in OWL – we will introduce a number of further expressive features later on. For now, however, we focus on the most basic aspects of OWL that do not depend on concrete features.

When speaking of an *(OWL) ontology* below, we always mean its formal, machine-readable content, which essentially is a set of OWL axioms. The informal documentation of the intended meaning is an important part of ontology engineering, but will not be discussed here. Information that is formally expressed in OWL can be used to draw interesting new conclusions, even without taking the informal meaning into account.

Example 2. If we happen to learn that a particular individual, say *Silvester*, is a cat, then from Example 1 above we can conclude that Silvester is also a mammal. In OWL, the fact that there is an individual in a class is expressed as follows:

ClassAssertion(*FelisCatus silvester*)

and, together with the axiom from Example 1, an OWL reasoner would now be able to draw the expected conclusion ClassAssertion(*Mammalia silvester*).

Reasoning (more precisely: *deductive* reasoning) is the task of computing such conclusions. The W3C standard defines which conclusions are entailed by an ontology, thus specifying the *semantics* of OWL. While this provides the official requirements for correct implementations, it does not explain how to actually compute the required inferences in practice. Indeed, there are many deduction methods for OWL, and the development of efficient approaches is an active area of research. We will look at various concrete algorithms in more detail later on.

Users of OWL can actually select between two slightly different semantics:

– The *Direct Semantics* defines the meaning of OWL axioms directly by relating them to *description logic* (DL), a fragment of first-order logic that provides similar expressive features as OWL.
– The *RDF-Based Semantics* first translates OWL axioms into directed graphs in the W3C's data exchange language *RDF*, where each axiom can lead to many edges (called *triples*) in the graph. The semantics is then defined for arbitrary RDF graphs, whether or not they can be obtained by translating an actual set of axioms.

Both semantics have advantages and disadvantages, and they can also lead to different conclusions. The Direct Semantics is not applicable to all RDF databases that use OWL features, whereas the RDF-Based Semantics does not allow for algorithms that compute all specified conclusions in all cases (i.e., reasoning is *undecidable*). In practice, however, both limitations are not as severe as one might expect. Direct Semantics tools can handle arbitrary RDF data using tolerant, "best effort" parsing. RDF-Based Semantics tools often specialise to decidable sublanguages (typically OWL RL) and ignore "unreasonable" combinations of features. In this chapter, we mainly use the Direct Semantics, since it is easier to explain, without having to introduce RDF first. This perspective is also closer to the view of ontology engineers, who normally design an ontology by editing OWL axioms, not RDF triples. However, actual reasoning algorithms for either semantics can be very similar, and lead to the same conclusions in many practical cases.

The above discussion also hints at the fact that there are many ways of writing OWL ontologies syntactically. The notation used in our above examples is known as the *Functional-Style Syntax* (FSS), since expressive features, such as SubClassOf, are written like function symbols in prefix notation. Two additional syntax standards for OWL axioms are *OWL/XML* and the *Manchester Syntax*. Moreover, OWL axioms can be faithfully represented in RDF graphs, which in turn can be written (i.e., *serialised*) in various syntactic forms, such as the *RDF/XML* syntax or the more concise *Turtle* syntax. Among those many options, FSS represents the data model of OWL most closely, whereas RDF/XML is the main exchange syntax that is most common on the Web. We will prefer FSS here since it is much more concise.[2] Moreover, we will soon introduce an even more concise writing based on description logics.

Summary The W3C OWL standard can represent machine-readable ontologies in a variety of syntactic encodings. Two semantics formally define the entailments of OWL.

2.2 OWL Reasoning Tasks

When deploying an OWL ontology in applications, explicitly stated axioms are just as relevant as the ones that are entailed. In other words, the *meaning* of an ontology is given by all the conclusions one can draw from it, no matter which of these conclusions are explicitly stated. Reasoning therefore is important for using and also for designing ontologies. Indeed, ontology engineers must be able to check the consequences of an ontology, just as as a software engineer must be able to test a program.

Every ontology (even if it is empty) entails an infinite number of OWL axioms. Therefore, the purpose of reasoning algorithms is generally not to compute all entailments, but merely all entailments *of a particular form*. This leads to various *reasoning tasks* that are of particular importance. This section gives an overview of the most common such tasks.

Example 2 illustrated a particular kind of deductive reasoning where we are interested in deriving a ClassAssertion axiom. The reasoning task of computing the individuals that belong to a given class (or set of classes) is called *instance retrieval*. If we merely want to find out whether one particular individual belongs to the given class, this task is called *instance checking*. Analogous tasks exist for SubClassOf axioms: computing all subclass relationships between a set of classes is called *classification*, and checking a particular subclass relationship is called *subsumption checking*.

[2] Contrary to popular belief, FSS is also easier to parse than RDF/XML, since the latter requires multiple passes to re-assemble OWL axioms from XML-encoded RDF triples. The practical importance of RDF/XML mainly stems from its wide use for encoding RDF data on the Web.

Another important reasoning task is *consistency checking*, the task of determining whether an ontology is logically *consistent* or *contradictory*. All OWL axioms in our previous examples required a particular relationship to hold, but axioms can also be used to state that a relationship must not hold, as illustrated in the next example.

Example 3. OWL allows us to express the *disjointness* of two classes, i.e., to say that two classes must never have any instances in common. For example, we can state that humans are distinct from cats using the axiom

DisjointClasses(*FelisCatus HomoSapiens*)

If an additional axiom ClassAssertion(*HomoSapiens silvester*) would now assert that Silvester is also human (in addition to him being a cat as stated in Example 2), then a reasoner would infer the ontology to be inconsistent.

According to (both of) the OWL semantics, an inconsistent ontology entails every axiom, and is therefore of no practical use. Inconsistency detection is thus important when developing ontologies. A closely related problem is *inconsistency* (or *incoherence*) of classes. A class is inconsistent if it is necessarily empty, i.e., if the ontology can only be consistent if the class contains no elements. For example, we could formalise an inconsistent class by requiring it to be disjoint with itself. Inconsistent classes are typically modelling errors, of which the ontology engineer should be alerted. Especially, OWL already includes a special class name owl:Nothing[3] to refer to an empty class, so it is never necessary to define additional inconsistent classes.

This completes our overview of the most important reasoning tasks. Luckily, many of these standard reasoning tasks can be solved by very similar algorithms. This is due to the following relationships:

- An ontology is inconsistent if some arbitrary class *SomeClass* that is not used in any axiom is inconsistent.
- A class *SomeClass* is inconsistent if the subsumption SubClassOf(*SomeClass* owl:Nothing) is entailed.
- A subsumption SubClassOf(*ClassA ClassB*) is entailed if the fact ClassAssertion(*ClassB something*) is entailed when extending the ontology with the axiom ClassAssertion(*ClassA something*), where *something* is a new individual name.
- A fact ClassAssertion(*SomeClass something*) is entailed if the ontology becomes inconsistent when adding the axioms DisjointClasses(*SomeClass AnotherClass*) and ClassAssertion(*AnotherClass something*), where *AnotherClass* is a new class not used anywhere yet.

This cyclic reduction allows us to reformulate each of the above reasoning problems in terms of any other, although we might need to modify the ontology for this purpose. Note that only very few features are needed for this reduction: all sublanguages of OWL that we consider in this chapter will thus allow us to do this. Instance retrieval and

[3] The actual name of owl:Nothing is http://www.w3.org/2002/07/owl#Nothing but we do not go into the details of class naming in OWL, and thus use a common abbreviation.

classification tasks can be solved by using many individual instance and subsumption checks. However, this is rarely the most efficient approach, and dedicated algorithms rather try to perform many instance or subsumption checks at once.

Summary The need to compute deductions for OWL ontologies leads to a number of standard reasoning tasks, which can be reduced to each other with only little effort.

2.3 Hardness of Reasoning

How difficult is it to actually compute entailments of OWL ontologies? To answer this question, we need to be more specific. Indeed, it is very easy to compute *some* OWL entailments – the challenge is to compute *all* entailments of a certain kind. Our requirements towards a "good" reasoner are usually as follows:

- *Soundness:* All computed inferences are really entailed.
- *Completeness:* All entailed inferences (of the kind we are interested in) are really computed.

The lack of completeness is sometimes accepted if it allows for simpler or more efficient implementations. This depends on the application: ontology engineers usually want to know all relevant consequences, while users of ontologies might already be content to retrieve some of the relevant information. Ideally, it should be clear and well-documented under which circumstances certain entailments will be missed, so that users can decide whether this is acceptable or not. The lack of soundness, in contrast, is usually not desirable, since unsound systems can only reliably tell us which axioms are *not* entailed, and this information is less interesting in most applications.

The other main requirement beyond soundness and completeness is efficiency: we want reasoning algorithms to use as little time and memory as possible. So our question should be: how hard is it to implement efficient, sound and complete reasoning procedures for OWL? A partial answer to this is given by computational complexity theory, which allows us to classify reasoning tasks according to their worst-case requirements in terms of time or memory. In their most common form, complexity measures refer to *decision problems*, i.e., to the complexity of solving problems that can only have either *yes* or *no* as an answer. Clearly, all the above checks for instances, subsumptions, and inconsistency are decision problems in this sense. The following can be stated:

- The standard entailment checks for OWL under RDF-Based Semantics are undecidable, i.e., there is no sound and complete reasoning algorithm that terminates in finite time.
- The standard entailment checks for OWL under Direct Semantics are N2ExpTime-complete, i.e., there are sound and complete, non-deterministic reasoning algorithms that terminate in doubly exponential time.

Recall that the RDF-Based Semantics covers a more general RDF-based language; the above undecidability result refers to arbitrary ontologies from this language. The N2ExpTime complexity for reasoning under Direct Semantics can roughly be described by saying: even if we are lucky enough to guess the right answer, it still takes doubly exponentially long to verify this guess. Recall that a double exponential function in n is linear in k^{k^n} for some constant k, which is not to be confused with $k^n \times k^n$ (e.g., $2^{2^4} = 65536$ while $2^4 \times 2^4 = 256$). Given that already NP algorithms (those that perform this check in polynomial time) are often considered infeasible in practice, this paints a rather gloomy picture indeed.

However, reasoning in practice rarely is close to this worst-case estimation. Implementations have been optimised to efficiently deal with typical ontologies, and often perform very well in these cases. Yet, the complexity result shows that it will always be possible to construct inputs that reasoners will fail to process (in practice, reasoners rarely succeed after running for a very long time; they rather run out of memory or need to use secondary memory that is so slow that processing must be aborted). Moreover, although the worst case is not typical in practice, even highly optimised implementations cannot be expected to scale up linearly to very large ontologies. What is worse, the performance of reasoners is very hard to predict, and small changes or errors may lead to significant differences.

The three profiles of OWL have been proposed as a way to overcome this problem. The high worst-case complexity of OWL is based on the interaction of many different expressive features. By restricting the supported features syntactically, each OWL profile defines a subset of ontologies for which standard reasoning tasks are *tractable*, i.e., for which reasoning is possible in polynomial time (or even less). In general, a *tractable ontology language* is one for which standard reasoning is tractable in this sense.

In practice, worst-case complexities can only provide partial information about the hardness of a problem. For example, an algorithm that solves a problem of size n in time n^{42} is polynomial, but it would still be infeasible to use it in practice. Indeed, even quadratic runtime behaviour is often not tolerable. Nevertheless, optimised reasoners for the OWL profiles have also been shown to achieve excellent performance in many practical cases. Moreover, algorithms that are worst-case polynomial are usually less complicated and easier to implement efficiently than algorithms of exponential complexity. This is certainly the case for the OWL profiles.

Summary Sound and complete OWL reasoning is of high complexity or even undecidable. The OWL profiles restrict OWL to improve complexity and practical performance.

2.4 Further Expressive Features in OWL

So far, we have only introduced a few expressive features of OWL by means of examples. In this section, we take a closer look at the basic expressive features of OWL, and introduce a slightly bigger set of basic features. The complete feature set of OWL is a lot bigger (see Section 5), but the features we select here are interesting enough to explain many important concepts related to OWL and its profiles.

OWL axioms are formed by combining *vocabulary entities* (e.g., *FelisCatus* or *silvester*) by means of *language constructors* for axioms and other expressions (e.g., SubClassOf and DisjointClasses). Vocabulary entities can have three different basic types:

– *Individual names* refer to individual objects.
– *Class names* refer to sets of objects.
– *Property names* refer to binary relationships between objects.

Example 4. We have already used the class name *FelisCatus* to represent the set of all cats, and the individual name *silvester* to represent Silvester the cat. For an example of a property, let *preysOn* express the relationship between a predator and its prey. For instance, we could state that Silvester preys on Tweety:

ObjectPropertyAssertion(*preysOn silvester tweety*)

It is customary to use lower case letters for the names of individuals and properties, and upper case letters for the names of classes, as we already did in all examples above.

Remark 5. OWL does not require entities to have a unique type. For example, we could use *FelisCatus* as a class name as above, and additionally use it as an individual name to assert that it is a species:

ClassAssertion(*Species FelisCatus*)

Under the Direct Semantics, this does not establish any formal relationship between *FelisCatus* the class and *FelisCatus* the individual. They still represent either an object or a set of objects, depending on the context they are used in, but never both at the same time. The use of the same name for two things is therefore known as *punning*. In spite of its weak semantics, it can still be convenient in ontological modelling.

Under the RDF-Based Semantics, the relationship is stronger and leads to additional semantic entailments. However, it is difficult to compute these entailments in all cases: it is one of the reasons why reasoning under RDF-Based Semantics is undecidable.

OWL provides many language constructors to express statements about vocabulary entities. So far, we have encountered examples of various types of axioms:

- SubClassOf: a class is a subclass of another
- DisjointClasses: two (or more) classes must not share elements
- ClassAssertion: a class contains an individual
- ObjectPropertyAssertion: a property relates two individuals

There are a number of further types of axioms in OWL. However, most of the expressivity of OWL is in its *class constructors* that allow us to build complex class descriptions from basic vocabulary entities. Since classes represent sets, the standard set operations are an obvious candidate for this:

- ObjectIntersectionOf: the intersection of two (or more) classes
- ObjectUnionOf: the union of two (or more) classes
- ObjectComplementOf: the complement of a class

Example 6. The following examples show how these constructors could be used:

- ObjectIntersectionOf(*FelisCatus Hungry*): the class of objects that are in the class *FelisCatus* and in the class *Hungry*
- ObjectUnionOf(*FelisCatus SerinusCanaria*): the class of objects that are cats (*FelisCatus*) or canary birds (*SerinusCanaria*)

– ObjectComplementOf(*HomoSapiens*): the class of objects that are not humans

Such class descriptions can be used in all OWL axioms that work with class names, e.g., to say that birds are not cats:

SubClassOf(*SerinusCanaria* ObjectComplementOf(*FelisCatus*))

or that Silvester is either a cat or a human (or both – this case is not excluded here):

ClassAssertion(*silvester* ObjectUnionOf(*FelisCatus HomoSapiens*))

Note that intersection, union, and complement therefore correspond to the logical operations of conjunction (and), disjunction (or), and negation (not). The empty class owl:Nothing that we encountered earlier can be viewed as a class constructor without arguments. Its dual is the class owl:Thing that contains all objects. The logical counterparts of owl:Nothing and owl:Thing would be constants *false* and *true*.

Remark 7. Many features of OWL have overlapping expressivity, and the same statement can usually be expressed in various ways. For example, the following five axioms are semantically equivalent:

```
DisjointClasses( FelisCatus HomoSapiens )
    SubClassOf( FelisCatus ObjectComplementOf( HomoSapiens ) )
    SubClassOf( HomoSapiens ObjectComplementOf( FelisCatus ) )
    SubClassOf( ObjectIntersectionOf( FelisCatus HomoSapiens ) owl:Nothing )
    SubClassOf( owl:Thing ObjectUnionOf( ObjectComplementOf( FelisCatus )
                                         ObjectComplementOf( HomoSapiens ) )
```

Therefore, DisjointClasses is not really needed and merely makes some statements more convenient. We call such features *syntactic sugar*.

The operations we have encountered so far are only related to basic Boolean operations of classes. In many cases, however, we would like to take properties into account when defining classes. This is possible using *property restrictions*. We will discuss two features of this type here: ObjectSomeValuesFrom and ObjectAllValuesFrom.

Example 8. In Example 4 we have stated that Silvester is preying on Tweety. Using property restrictions, we can define the class of all objects that prey on some canary bird:

ObjectSomeValuesFrom(*preysOn SerinusCanaria*)

Note that this is not an axiom but merely a class expression. A predator can be described as something that preys on anything (not just on canary birds). The class owl:Thing is handy for relaxing the description accordingly:

SubClassOf(ObjectSomeValuesFrom(*preysOn* owl:Thing) *Predator*)

In other words, ObjectSomeValuesFrom expresses an *existential restriction* that refers to the existence a certain property relationship. Dually, ObjectAllValuesFrom encodes a *universal restriction* that refers to all relationships of a property:

Example 9. The following defines the class of objects for which all *preysOn* relations point to some canary bird, i.e., the class of the things that prey on canary birds only:

ObjectAllValuesFrom(*preysOn SerinusCanaria*)

This can be used, e.g., to state that one can only prey on animals. In this case, we use the class owl:Thing to state that something is true for all things:

SubClassOf(owl:Thing ObjectAllValuesFrom(*preysOn Animal*))

This yields a *property range axiom*, stating that everything that somebody preys on must be an animal. Importantly, this does not mean that every object preys on something in the first place: if something has no *preysOn* relations, then all of its (zero) *preysOn* relations satisfy the requirement. This is the usual reading of universal quantifiers in logic, but it can be a source of confusion. In daily live, statements like “all of my children have won the Nobel prize” are suggesting that at least one such child exists.

This completes the set of class constructors that we want to consider here, although there are a number of further features of this kind. In contrast, expressions for constructing complex properties are much less frequent in OWL. In fact, the only such constructor in OWL is ObjectInverseOf, which allows us to reverse the direction of a property:

Example 10. The inverse of *preysOn* can be described in OWL as ObjectInverseOf(*preysOn*); it is the property relating a prey to its predator. One could use it, e.g., to describe the class of objects that are preyed on by some cat:

ObjectSomeValuesFrom(ObjectInverseOf(*preysOn*) *FelisCatus*)

Using inverses, the property range axiom of Example 9 could also be written as

SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(*preysOn*) owl:Thing) *Animal*)

which again can be read as “everything that is preyed on by anybody must be an animal.”

Summary Statements in OWL are built from a vocabulary of individual, class, and property names using various constructors for axioms, and class and property expressions.

2.5 From OWL to Description Logics

OWL is closely related to (and partly based on) a family of knowledge representation languages called *description logics* (DLs). DLs have inspired many of the current expressive features of OWL, and they are the basis for defining OWL’s Direct Semantics.

Table 1. Translating OWL expressions to description logics

	OWL Functional-Style Syntax	DL Syntax
Axioms	SubClassOf(C D)	$C \sqsubseteq D$
	ClassAssertion(C a)	$C(a)$
	ObjectPropertyAssertion(P a b)	$P(a,b)$
Class expressions	ObjectIntersectionOf(C D)	$C \sqcap D$
	ObjectUnionOf(C D)	$C \sqcup D$
	ObjectComplementOf(C)	$\neg C$
	owl:Thing	$\top$
	owl:Nothing	$\bot$
	ObjectSomeValuesFrom(P C)	$\exists P.C$
	ObjectAllValuesFrom(P C)	$\forall P.C$
Property expressions	ObjectInverseOf(P)	P^-

A typical DL is a fragment of first-order predicate logic, and OWL reasoning under Direct Semantics can therefore be viewed as a special case of first-order logic reasoning. In addition, DLs come with a very convenient syntax for writing OWL axioms in much less space. In this section, we briefly introduce description logics from the perspective of OWL.

The building blocks of DL are very similar to that of OWL. DL axioms are constructed from vocabulary elements and various constructors (i.e., logical operators). Ontologies in DL are usually called *knowledge bases*; classes and properties in DL are called *concepts* and *roles*. To avoid confusion, however, we will stick to the OWL terminology throughout this chapter. The OWL axioms and expressions that we have encountered so far can easily be written in description logics according to Table 1.

Example 11. The following ontology illustrates some expressive features of DL:

$\mathit{FelisCatus}(\mathit{silvester})$	Silvester is a cat.	(1)
$\mathit{preysOn}(\mathit{silvester}, \mathit{tweety})$	Silvester preys on Tweety.	(2)
$\mathit{FelisCatus} \sqsubseteq \mathit{Mammalia}$	Cats are mammals.	(3)
$\exists \mathit{preysOn}.\top \sqsubseteq \mathit{Predator}$	What preys on something is a predator.	(4)
$\top \sqsubseteq \forall \mathit{preysOn}.\mathit{Animal}$	What is preyed on is an animal.	(5)
$\mathit{Animal} \sqcap \mathit{PlaysChess} \sqsubseteq \mathit{HomoSapiens}$	All animals that play chess are humans.	(6)
$\mathit{Mammalia} \sqsubseteq \exists \mathit{hasFather}.\mathit{Mammalia}$	Every mammal has a mammal father.	(7)

Note that, intuitively speaking, the meaning of $\sqcap$, $\sqcup$, and $\sqsubseteq$ corresponds to the well-known set theoretic operations and relations $\cap$, $\cup$, and $\subseteq$. We will see in Section 2.6 below that this intuition also agrees with the precise definition of the DL semantics.

As in the case of OWL, one can obtain different DLs by adding or omitting expressive features. The description logic that supports all class expressions with $\top$, $\bot$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, and $\forall$ is known as $\mathcal{ALC}$ (which originally used to be an abbreviation for *Attribute Language with Complement*). Inverse properties are not supported by $\mathcal{ALC}$, and the DL we have introduced above is actually called $\mathcal{ALCI}$ (for $\mathcal{ALC}$ *with inverses*). Many description logics can be defined by simply listing their supported features, but there are also cases where some features are only allowed in certain places (we will see examples of this in the profiles OWL RL and OWL QL later on). It is therefore easy to obtain a big number of different DLs, and many of them have been named and studied. The motivation for considering that many different DLs is that even slight changes in the supported features can lead to very different computational properties. In general, the more features we allow, the more complex and complicated reasoning becomes. On the other hand, the following example shows that the omission of a (syntactic) feature does not necessarily reduce expressivity, since we might still be able to express the same thing indirectly.

Example 12. The OWL axiom DisjointClasses(*FelisCatus HomoSapiens*) cannot be directly expressed in DLs, but we can easily encode it using any of the following four axioms (see also Remark 7):

$$\begin{aligned}
\mathit{FelisCatus} &\sqsubseteq \neg \mathit{HomoSapiens}\\
\mathit{HomoSapiens} &\sqsubseteq \neg \mathit{FelisCatus}\\
\mathit{FelisCatus} \sqcap \mathit{HomoSapiens} &\sqsubseteq \bot\\
\top &\sqsubseteq \neg \mathit{FelisCatus} \sqcup \neg \mathit{HomoSapiens}
\end{aligned}$$

Comparing this with Remark 7, it is evident that the DL notation saves a lot of space. By discarding syntactic sugar such as DisjointClasses, we can also reduce the number of cases that we need to consider in definitions and algorithms, which will generally simplify the presentation.

DL ontologies are often structured into two sets: *ABox* and *TBox*. The ABox consists of all (class or property) assertions. The TBox consists of all *terminological* axioms, i.e., of all subclass inclusion axioms. The ABox provides information about concrete individuals while the TBox describes general rules that hold for all individuals. In consequence, ABoxes tend to be much larger than TBoxes. Moreover, TBoxes are usually more general (e.g., a biological taxonomy of species) while ABoxes are specific to an application (e.g., a database of a zoo that stores the species of each of its animals). Ontology engineers often develop TBoxes, which users of ontologies combine with existing ABox information (that might also come from traditional databases). In spite of these practical differences, TBox and ABox axioms can be treated in mostly the same way when explaining the underlying theory.

Summary Description logics provide a concise language for OWL axioms and expressions. DLs are characterised by their expressive features. The DL we use here is $\mathcal{ALCI}$.

2.6 The OWL Direct Semantics

For following the rest of this chapter, it will suffice to understand the intuitive semantics of each of the above description logic constructs (and the corresponding OWL expressions). When developing reasoning algorithms, however, a more precise definition of

Table 2. Interpreting description logic expressions semantically

	Expression *ex*	Interpretation $ex^{\mathcal{I}}$
Class expressions	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
	$\top$	$\Delta^{\mathcal{I}}$
	$\bot$	$\emptyset$
	$\exists P.C$	$\{e \mid \text{there is } f \text{ with } \langle e, f\rangle \in P^{\mathcal{I}} \text{ and } f \in C^{\mathcal{I}}\}$
	$\forall P.C$	$\{e \mid \text{for all } f \text{ with } \langle e, f\rangle \in P^{\mathcal{I}} \text{ we have } f \in C^{\mathcal{I}}\}$
Property expressions	P^{-}	$\{\langle f, e\rangle \mid \langle e, f\rangle \in P^{\mathcal{I}}\}$

the semantics is needed to check if the algorithm really computes the right results. In this section, we will therefore define the semantics of DL, which is known as the *Direct Semantics* of OWL. This section and the next could be skipped by readers who are interested in a general overview only.

The semantics of DL is based on the simple idea that every ontology specifies information about a possible "state of the world." Initially, all that we know is that individual names represent objects, classes represent sets of objects, and properties represent binary relations between objects. Many different situations are possible: we do not know which objects, sets, or relations our vocabulary symbols are meant to represent. By asserting a DL axiom, we narrow down the possibilities. For example, the axiom $\mathit{FelisCatus} \sqsubseteq \mathit{Mammalia}$ can only be satisfied if the set represented by *FelisCatus* is a subset of the set represented by *Mammalia*, even if we still have infinitely many possible ways to interpret these sets. Adding more axioms will further restrict the possible interpretations (i.e., "states of the world"). An entailment of an ontology then is simply an axiom that is satisfied by every interpretation that satisfies all axioms of the ontology.

To make this idea formal, we need to define what an *interpretation* really is, mathematically speaking, and which conditions need to be met by it in order to satisfy a particular axiom. In other words, we need to define a *model theory*. This approach is similar to first-order logic, but our interpretations can be a bit simpler due to the restrictions imposed by DL.

Definition 13. An *interpretation* $\mathcal{I}$ consist of a non-empty set $\Delta^{\mathcal{I}}$ (called its *domain*), and an interpretation function $\cdot^{\mathcal{I}}$ that assigns

- every individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- every class name C to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- every property name P to a binary relation $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

This definition merely formalises what we have said above: individuals are objects, classes are sets, and properties are relations. Once we have such an interpretation of the basic vocabulary, the meaning of class and property expressions, and the truth of axioms can be calculated.

Definition 14. The value of an interpretation $\mathcal{I}$ for class and property expressions is defined as in Table 2. A DL axiom *ax* is *satisfied* by $\mathcal{I}$, written $\mathcal{I} \models ax$, if the corresponding condition holds:

- $\mathcal{I} \models C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $\mathcal{I} \models C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$,
- $\mathcal{I} \models P(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in P^{\mathcal{I}}$.

$\mathcal{I}$ satisfies an ontology O, written $\mathcal{I} \models O$, if it satisfies all axioms in O. In this case, $\mathcal{I}$ is called a *model* of O.

Example 15. Consider the interpretation $\mathcal{I}$ with domain $\Delta^{\mathcal{I}} = \{\heartsuit, \clubsuit, \star\}$, and the following interpretations for vocabulary symbols:

$\mathit{silvester}^{\mathcal{I}} = \clubsuit$	$\mathit{tweety}^{\mathcal{I}} = \heartsuit$
$\mathit{FelisCatus}^{\mathcal{I}} = \{\clubsuit\}$	$\mathit{preysOn}^{\mathcal{I}} = \{\langle\clubsuit, \heartsuit\rangle, \langle\star, \star\rangle\}$
$\mathit{Mammalia}^{\mathcal{I}} = \{\clubsuit, \star\}$	$\mathit{Predator}^{\mathcal{I}} = \{\clubsuit, \star\}$
$\mathit{Animal}^{\mathcal{I}} = \{\star, \heartsuit\}$	$\mathit{PlaysChess}^{\mathcal{I}} = \{\star\}$
$\mathit{hasFather}^{\mathcal{I}} = \{\langle\clubsuit, \star\rangle, \langle\star, \star\rangle\}$	$\mathit{HomoSapiens}^{\mathcal{I}} = \{\}$

This interprets all vocabulary symbols that occur in Example 11. It is easy to check that axiom (6) is the only axiom of that example that is not satisfied by $\mathcal{I}$. Indeed, $\star \in (\mathit{Animal} \sqcap \mathit{PlaysChess})^{\mathcal{I}}$ but $\star \notin \mathit{HomoSapiens}$.

Note that interpretations do usually not fully capture our intuition about the domain that we model. For example, the interpretation asserts that $\star$ is its own father – none of our axioms state that this should not be possible. Moreover, we might be surprised that $\mathit{Mammalia}^{\mathcal{I}} \not\subseteq \mathit{Animal}^{\mathcal{I}}$ and that there are no humans in our model. It is usually impossible to fully enforce one particular interpretation, and it is also unnecessary, since we are interested in the logical conclusions rather than in the exact shape of possible models. Indeed, modelling too many details can also slow down reasoning without leading to any new consequences that are relevant to the application. The challenge in modelling ontologies therefore is to understand to which extent the models can and should mirror reality.

Definition 16. A DL ontology O *entails* a DL axiom *ax*, written $O \models ax$, if every model of O is also a model of *ax*. O is *inconsistent* if it has no models. O entails that a class C is *inconsistent* if $C^{\mathcal{I}} = \emptyset$ in all models $\mathcal{I}$ of O.

An OWL axiom is entailed by an OWL ontology under Direct Semantics if the corresponding DL axiom is entailed by the corresponding DL ontology. Inconsistency of OWL ontologies and classes under Direct Semantics is defined analogously.

Note that this standard definition of entailment means that inconsistent ontologies entail every axiom. Indeed, each of the (zero) models of an inconsistent ontology satisfies any axiom. Inconsistency of ontologies and classes can be expressed as axiom entailment problems:

- O is inconsistent exactly if $O \models \top \sqsubseteq \bot$.
- O entails a class C to be inconsistent exactly if $O \models C \sqsubseteq \bot$.

Summary Entailments of DL ontologies are defined with a model theoretic semantics by interpreting individual names as objects, classes as sets, and properties as relations. By translating OWL to DL, this also provides the Direct Semantics for OWL.

2.7 The OWL RDF-Based Semantics

As explained above, the RDF-Based Semantics of OWL is not based on description logics. It uses a similar model theoretic approach, but interpretations are based on the graph-based data model of RDF, which is not the topic of this chapter. Nevertheless, the underlying intuitive interpretation of individuals, classes, and properties is the same as in the case of DL and the Direct Semantics. Fortunately, this similarity extends to the formal semantics, and many entailments under RDF-Based Semantics can also be obtained under Direct Semantics. This section explains the practical impact of this relationship.

The main ingredient of our discussion is the following *correspondence theorem*, which we will discuss in detail below. This theorem is derived from a more general correspondence theorem in the OWL 2 standard, where a full proof can be found [40, Section 7.2].

Theorem 17. Let O be an OWL ontology and let *ax* be an OWL axiom that contains only individuals, class expressions, and property expressions that also occur in O. If O entails *ax* under Direct Semantics, then O entails *ax* under RDF-Based Semantics.

Moreover, if O is inconsistent under the Direct Semantics, then O is also inconsistent under the RDF-Based Semantics. In this case, O entails every axiom under either semantics.

Let us first discuss the significance of this theorem. In essence, it states that, under certain conditions, entailments under Direct Semantics are also valid under RDF-Based Semantics. In practice, this means that we can use a Direct Semantics algorithm to build a sound but incomplete reasoner for RDF-Based Semantics. Considering the fact that there can be no sound and complete, terminating algorithm for the RDF-Based Semantics (since it is undecidable), this is actually not too bad. However, in many cases one could obtain a "more complete" procedure by taking additional aspects of the RDF-Based Semantics into account. Even if this is done, the actual reasoning algorithms could use similar principles, and might be faced with similar implementation challenges. From the viewpoint of a practitioner, the relevance of this correspondence strongly depends on two further questions:

(1) Is the theorem applicable in many practical cases?
(2) Are the RDF-based entailments that we can obtain from the Direct Semantics enough for being useful in practice?

Question (1) essentially asks if the requirements of Theorem 17 are too restrictive to apply it in realistic cases. We require that individuals, classes, and properties in *ax* must occur in O. From the viewpoint of Direct Semantics (and DLs), this turns out to be a very weak (and even strange) condition. For example, one could ensure that O contains C by adding an axiom SubClassOf(C owl:Thing) (in DL terms: $C \sqsubseteq \top$). This does not change the meaning under Direct Semantics: we simply have stated that all objects in the class C are contained in the domain of interpretation. Similar *tautological* axioms can be used to introduce arbitrary individual names and property expressions. Therefore, every OWL ontology can be extended to satisfy the preconditions of Theorem 17 without changing its Direct Semantics.

Under RDF-Based Semantics, however, such additional axioms do make a difference: it only supports entailments about entities that are syntactically present in the ontology. On the other hand, many natural reasoning tasks relate to expressions that occur in the ontology. The second part of the theorem illustrates that inconsistency checking is also very similar under both semantics, even without additional side conditions. In summary, we find that Theorem 17 is applicable in many practical situations.

There is another "hidden" precondition in Theorem 17 that should be noted. Namely, we start from the assumption that ontologies are given as sets of OWL axioms. The RDF-Based Semantics, in contrast, can also be applied to arbitrary RDF graphs that may not correspond to any set of OWL axioms in a clean way. In other words, the theorem restricts to the part of RDF-based OWL ontologies that can be represented in terms of OWL axioms. Thus, if we want to process arbitrary RDF documents under RDF-Based Semantics, there is a second source of incompleteness, since ignoring some of the input (the part that does not represent OWL axioms) may lead us to fewer conclusions.[4] Hence, this aspect is more closely related to our second question.

Question (2) can be rephrased as follows: if a Direct Semantics reasoner is incomplete for RDF-Based Semantics, how many practically interesting conclusions will it be missing? We cannot expect a quantitative answer (such as "85.3% of all entailments are found"), because there are infinitely many entailments under either semantics. In general, it is difficult to measure the degree of incompleteness. However, even a coarse answer to the question is difficult to give, partly due to the fact that there has not been much research on this topic. We already mentioned that some RDF documents cannot be mapped to OWL ontologies without loosing some information, and it is clear that in such cases we can expect additional entailments under RDF-Based Semantics. Moreover, even valid OWL ontologies contain some information that is not taken into account when reasoning under the Direct Semantics. For example, every OWL ontology has an *ontology header* that can be used to specify, e.g., the name of the ontology. Under RDF-Based Semantics, this information is used during reasoning (e.g., the ontology name is treated in the same way as Silvester the cat, and both might be inferred to occur in classes and properties). Again, it is obvious and well-understood that this will lead to conclusions that we cannot obtain under the Direct Semantics. Therefore the essential remaining question is:

> Is there an OWL ontology O and an axiom *ax* that has a logical meaning in DL (i.e., that is not something like the ontology header), such that O entails *ax* under the RDF-Based Semantics but not under the Direct Semantics?

In general, the answer to this question is *yes*. An example is given in Remark 18 below, which possibly covers the most important practical difference between the two seman-

[4] Fortunately, disregarding part of the input can never lead to *more* conclusions. In other words, the RDF-Based Semantics is *monotone*, just like the Direct Semantics.

tics. However, this example uses some features that we have not introduced so far. It is an open question if there is any such case if we restrict to axioms that are expressible in $\mathcal{ALCI}$. In fact, it is conceivable that the logical entailments under Direct Semantics and RDF-Based Semantics are exactly the same for an interesting part of OWL.

Remark 18. The most prominent example where OWL's RDF-Based Semantics is notably different from the Direct Semantics is related to equality. In OWL, it is possible to state that two individuals are equal, i.e., that two different individual names refer to the same object. As explained in Remark 5, we can also do this for entities that represent classes in other contexts, e.g., to state that the species *Felis catus* is the same as *housecat*:

SameIndividual(*FelisCatus Housecat*)

In the Direct Semantics, this only means that the *individuals* represented by these names are equal, but it would not imply anything about the classes of the same name. In the RDF-Based Semantics, in contrast, classes are identified with individuals, and the above statement would imply, e.g., that every member of the species *Felis catus* is also a housecat:

SubClassOf(*FelisCatus Housecat*)

This would not follow from the Direct Semantics. However, it would be easy to add this RDF-based entailment to a Direct Semantics reasoner by silently adding, for every equality assertion between two individuals, two mutual subclass inclusion axioms (and, to cover the case that equal entities are also used as properties, additional property inclusion axioms; we have not discussed this here).

The situation becomes more complicated if the equality of two individuals is not asserted but entailed indirectly. Again, this is not possible when restricting to the features we have discussed so far, but it can happen when using additional OWL features. In general, the interpretation of inferred equalities as subclass relationships makes reasoning undecidable, but there are cases (notably OWL RL) where inferred equalities can still be treated like asserted ones.

In practice, covering asserted equalities can be very important since users may (intentionally or accidentally) use SameIndividual to state that two classes or two properties are the same. Inferred equalities between class or property names, in contrast, should rarely have that practical importance.

Summary OWL's RDF-Based Semantics is based on a model theory that is defined on RDF documents. Yet, many entailments agree with those under the Direct Semantics.

3 Reasoning in the OWL Profiles

Equipped with basic background knowledge about OWL and its semantics, we are now ready to discuss the lightweight profiles of OWL. We first define small sublanguages of OWL that we use for explaining the core ideas underlying each profile, and then discuss typical reasoning algorithms for each language that illustrate how many practical systems operate.

Throughout this section, we use the DL syntax introduced above for its brevity. It should be understood as an abbreviation for OWL axioms, that does not necessarily require Direct Semantics to be used for defining entailments. Indeed, the algorithms that we discuss are also applicable to reasoning under RDF-Based Semantics, as shown in Theorem 17.

3.1 Three Tiny OWLs

To explain the characteristics of the three OWL profiles, we introduce three small ontology languages $\mathcal{EL}\mathit{tiny}$, $\mathcal{RL}\mathit{tiny}$, and $\mathcal{QL}\mathit{tiny}$ that represent typical features of the profiles. To do this, let us first sum up the features of $\mathcal{ALCI}$. An easy way to do this is to describe the syntax using formal grammars. The following language **Axiom** describes the three possible forms of $\mathcal{ALCI}$ axioms, based on some auxiliary languages for class and property expressions:

$\mathcal{ALCI}$

$$\begin{aligned}
\textbf{Axiom} &::= \textbf{C} \sqsubseteq \textbf{C} \mid \textbf{C}(\textbf{IName}) \mid \textbf{P}(\textbf{IName}, \textbf{IName}) \\
\textbf{C} &::= \textbf{CName} \mid \top \mid \bot \mid \textbf{C} \sqcap \textbf{C} \mid \textbf{C} \sqcup \textbf{C} \mid \neg\textbf{C} \mid \exists\textbf{P}.\textbf{C} \mid \forall\textbf{P}.\textbf{C} \\
\textbf{P} &::= \textbf{PName} \mid \textbf{PName}^{-}
\end{aligned}$$

Here and in the following, we use **IName**, **CName**, and **PName** for the sets of individual names, class names, and property names in our vocabulary. Recall that an expression like $\textbf{C} \sqsubseteq \textbf{C}$ represents any axiom of the form $D \sqsubseteq E$ with $D, E \in \textbf{C}$, including the case where $D \neq E$.

The three profiles of OWL are defined by restricting the features of $\mathcal{ALCI}$ appropriately. We first introduce $\mathcal{EL}\mathit{tiny}$, the language related to OWL EL, since it is easiest to define. It can be obtained by restricting $\mathcal{ALCI}$ to allow only conjunction, existential restrictions, top and bottom:

$\mathcal{EL}\mathit{tiny}$

$$\begin{aligned}
\textbf{Axiom} &::= \textbf{C} \sqsubseteq \textbf{C} \mid \textbf{C}(\textbf{IName}) \mid \textbf{P}(\textbf{IName}, \textbf{IName}) \\
\textbf{C} &::= \textbf{CName} \mid \top \mid \bot \mid \textbf{C} \sqcap \textbf{C} \mid \exists\textbf{P}.\textbf{C} \\
\textbf{P} &::= \textbf{PName}
\end{aligned}$$

This ontology language is very similar to a lightweight description logic known as $\mathcal{EL}$ (the only difference is that $\mathcal{EL}$ does not allow $\top$). This relationship is also the reason why the according OWL profile has been called EL. OWL EL actually also supports additional $\mathcal{ALCI}$ axioms of the form $\top \sqsubseteq \forall P.C$ (i.e., *property range axioms* as discussed in Example 9), but no other uses of universal quantifiers. We exclude this special case here for simplicity.

Example 19. The following is an example of an $\mathcal{EL}\mathit{tiny}$ ontology:

$$\mathit{FelisCatus} \sqsubseteq \exists \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small}) \tag{8}$$
$$\mathit{Animal} \sqcap \exists \mathit{preysOn}.\mathit{Animal} \sqsubseteq \mathit{Predator} \tag{9}$$
$$\mathit{FelisCatus} \sqsubseteq \mathit{Animal} \tag{10}$$

where the axioms state that every cat preys on some animal that is small (8), every animal that preys on some animal is a predator (9), and cats are animals (10).

When compared to other lightweight ontology languages, the characteristic feature of $\mathcal{EL}\mathit{tiny}$ is that it allows for arbitrary existential quantifiers but not for universal quantifiers. As shown in Example 10, inverse properties would be a loophole in this restriction, so they must be forbidden as well. Moreover, all tractable ontology languages disallow or restrict the use of union $\sqcup$, since this could otherwise require non-deterministic choices during reasoning (see Section 4.3). Complement $\neg$ must therefore also be restricted; otherwise one could simply express $C \sqcup D$ by writing $\neg(\neg C \sqcap \neg D)$. We will see other approaches to restrict $\sqcup$ and $\neg$ in $\mathcal{RL}\mathit{tiny}$ and $\mathcal{QL}\mathit{tiny}$.

The most typical usage of OWL EL in practice is the modelling of large biomedical ontologies. Such ontologies are carefully modelled by a group of experts to capture general domain knowledge (e.g., the fact that all cats are mammals). Existential quantifiers can be useful for defining general concepts, e.g., to express that a heart disease is a disease that occurs in *some* part of the heart.

The axioms of $\mathcal{EL}\mathit{tiny}$ are exactly those $\mathcal{ALCI}$ axioms that use only a certain subset of constructors ($\sqcap$, $\exists$, $\bot$, $\top$). The language $\mathcal{RL}\mathit{tiny}$ takes a slightly different approach for restricting $\mathcal{ALCI}$. Whether a constructor is allowed or not now depends on its position within an axiom. Roughly speaking, the constructors that are allowed on the left-hand side of class inclusions are different from those that are allowed on the right-hand side. We can capture this by defining two languages of class expressions as follows:

$\mathcal{RL}\mathit{tiny}$

$$\begin{aligned}
\mathbf{Axiom} &::= \mathbf{CL} \sqsubseteq \mathbf{CR} \mid \mathbf{CR}(\mathbf{IName}) \mid \mathbf{P}(\mathbf{IName}, \mathbf{IName})\\
\mathbf{CL} &::= \mathbf{CName} \mid \bot \mid \mathbf{CL} \sqcap \mathbf{CL} \mid \mathbf{CL} \sqcup \mathbf{CL} \mid \exists \mathbf{P}.\mathbf{CL}\\
\mathbf{CR} &::= \mathbf{CName} \mid \bot \mid \mathbf{CR} \sqcap \mathbf{CR} \mid \neg \mathbf{CL} \mid \forall \mathbf{P}.\mathbf{CR}\\
\mathbf{P} &::= \mathbf{PName} \mid \mathbf{PName}^{-}
\end{aligned}$$

Hence, for example, the axiom $C \sqcup D \sqsubseteq E$ is in $\mathcal{RL}\mathit{tiny}$, but $E \sqsubseteq C \sqcup D$ is not. Indeed, the meaning of $C \sqcup D \sqsubseteq E$ could also be captured using two axioms $C \sqsubseteq E$ and $D \sqsubseteq E$, so we can see that the union on the left of class inclusions does not really add expressive power. The situation for complement is similar: $C \sqsubseteq \neg D$ can be expressed as $C \sqcap D \sqsubseteq \bot$, as we already observed in Example 12. This explains why negated classes on the right-hand side are of the form $\neg\mathbf{CL}$ instead of $\neg\mathbf{CR}$. Therefore, union and complement in

$\mathcal{RL}\mathit{tiny}$ are mainly syntactic sugar that one could add to $\mathcal{EL}\mathit{tiny}$ as well (but at the cost of requiring separate left and right class expression languages, which are not otherwise needed in $\mathcal{EL}\mathit{tiny}$).

The characteristic feature of $\mathcal{RL}\mathit{tiny}$ is its asymmetric use of quantifiers: existentials are allowed only on the left and universals are allowed only on the right. Inverse properties can be allowed without weakening these restrictions. Also note that $\mathcal{RL}\mathit{tiny}$ does not allow $\top$.[5]

Example 20. The following is an example of an $\mathcal{RL}\mathit{tiny}$ ontology:

$$\mathit{FelisCatus} \sqsubseteq \forall \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small}) \tag{11}$$

$$\mathit{Animal} \sqcap \exists \mathit{preysOn}.\mathit{Animal} \sqsubseteq \mathit{Predator} \tag{12}$$

$$\mathit{FelisCatus} \sqsubseteq \mathit{Animal} \tag{13}$$

where axiom (11) states that cats prey only on small animals. Axioms (12) and (13) are the same as in Example 19.

The name *RL* hints at the fact that the restrictions lead to a kind of *rule language*, where all axioms can be expressed as rules (logical implications). For instance, axioms of Example 20 could be expressed as follows:

$$\mathit{FelisCatus}(x) \land \mathit{preysOn}(x, y) \to \mathit{Animal}(y) \land \mathit{Small}(y) \tag{14}$$

$$\mathit{Animal}(x) \land \mathit{preysOn}(x, y) \land \mathit{Animal}(y) \to \mathit{Predator}(x) \tag{15}$$

$$\mathit{FelisCatus}(x) \to \mathit{Animal}(x) \tag{16}$$

If we read this as first-order logic implications where all variables are universally quantified, then these rule capture exactly the Direct Semantics of the example ontology. The RDF-Based Semantics is not fully defined in this way, but the rule-based forms can still be used to draw valid conclusions in this case.

The ontology language $\mathcal{QL}\mathit{tiny}$ is defined in a similar way to $\mathcal{RL}\mathit{tiny}$ using separate grammars for class expressions on the left and right of class inclusion axioms:

$\mathcal{QL}\mathit{tiny}$

$$\begin{aligned}
\textbf{Axiom} &::= \textbf{CL} \sqsubseteq \textbf{CR} \mid \textbf{CR}(\textbf{IName}) \mid \textbf{P}(\textbf{IName}, \textbf{IName})\\
\textbf{CL} &::= \textbf{CName} \mid \top \mid \bot \mid \exists \textbf{P}.\top\\
\textbf{CR} &::= \textbf{CName} \mid \top \mid \bot \mid \textbf{CR} \sqcap \textbf{CR} \mid \neg\textbf{CL} \mid \exists \textbf{P}.\textbf{CR}\\
\textbf{P} &::= \textbf{PName} \mid \textbf{PName}^{-}
\end{aligned}$$

Notably, $\mathcal{QL}\mathit{tiny}$ is a lot more restricted than $\mathcal{EL}\mathit{tiny}$ and $\mathcal{RL}\mathit{tiny}$ regarding its Boolean constructors: it does not even allow intersection $\sqcap$ on the left-hand side. The

[5] This restriction of OWL RL does not have a deeper technical reason but is based on the preferences of OWL RL tool developers who participated in the W3C OWL Working Group.

reasons for this seem to be mostly historic – at least there is no big computational challenge in allowing $\sqcap$ as in $\mathcal{EL}$*tiny* and $\mathcal{RL}$*tiny*. The left-hand side also is restricted to a very specific form of existential restriction, where only $\top$ can be used as a filler; as opposed to the case of $\sqcap$, this restriction is very important for reasoning in $\mathcal{QL}$*tiny*. On the other hand, $\mathcal{QL}$*tiny* supports inverse properties (which are not in $\mathcal{EL}$*tiny*) and existential quantifiers on the right-hand side (which are not in $\mathcal{RL}$*tiny*).

Example 21. The following is an example of an $\mathcal{QL}$*tiny* ontology:

$$\begin{aligned} \mathit{FelisCatus} &\sqsubseteq \exists \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small}) && (17)\\ \exists \mathit{preysOn}.\top &\sqsubseteq \mathit{Predator} && (18)\\ \exists \mathit{preysOn}^{-}.\top &\sqsubseteq \mathit{Animal} && (19)\\ \mathit{FelisCatus} &\sqsubseteq \mathit{Animal} && (20) \end{aligned}$$

As before, we can state that every cat preys on some small animal (17), and that cats are animals (20). We can also say that everything that preys on anything else is a predator (18), without being able to specify the additional restriction to animals as in (9) above. On the other hand, we can state that everything that is preyed on must be an animal (19), which expresses a range axiom using inverses as in Example 10.

The specific feature combination supported by $\mathcal{QL}$*tiny* is motivated by its intended usage as a rich *query language* (hence the name *QL*). In *ontology-based data access* (OBDA), an ontology is used to augment an existing database (viewed as a set of facts/ABox). A (database) query then should return not only the data that is stored explicitly in the database, but also additional facts that can be inferred from it using the ontological information. OWL QL (and $\mathcal{QL}$*tiny*) allows this to be done without needing to write inferred facts to the database, but rather by using a query rewriting approach that we will explain in Section 3.8.

Summary The characteristics of the OWL profiles are exemplified by three languages: $\mathcal{EL}$*tiny* that allows $\exists$ but no $\forall$, $\mathcal{RL}$*tiny* that allows $\exists$ on the left and $\forall$ on the right of class inclusions, and $\mathcal{QL}$*tiny* that even restricts $\sqcap$ but allows $\exists$ and inverse properties.

3.2 Rule-based Reasoning for Instance Retrieval in $\mathcal{RL}$*tiny*

The first reasoning method that we present for the OWL profiles is for instance retrieval in OWL RL, that is, in our little fragment $\mathcal{RL}$*tiny* of OWL RL. The common approach to RL reasoning is to use a set of *inference rules* that are applied to an input ontology to derive new consequences. This is a fundamental technique that we will also encounter for $\mathcal{EL}$*tiny*, and we will use the example of $\mathcal{RL}$*tiny* to introduce related concepts that will also come in handy later on.

Instance retrieval is the most important inference task for OWL RL. Applications of RL often involve a large amount of explicit facts, which are augmented by a set of TBox axioms that is much smaller (typically by at least one order of magnitude). This situation is common for input ontologies that are obtained by crawling the Semantic

$$\mathbf{A}_{\sqsubseteq}\ \frac{D(c)}{E(c)} : D \sqsubseteq E \in O$$

$$\mathbf{A}^-_{\sqcap}\ \frac{D_1 \sqcap D_2(c)}{D_1(c) \quad D_2(c)} \qquad \mathbf{A}^+_{\sqcap}\ \frac{D_1(c) \quad D_2(c)}{D_1 \sqcap D_2(c)} : D_1 \sqcap D_2 \text{ occurs in } O$$

$$\mathbf{A}^-_{\forall}\ \frac{\forall P.E(c) \quad P(c,d)}{E(d)} \qquad \mathbf{A}^+_{\exists}\ \frac{P(c,d) \quad E(d)}{\exists P.E(c)} : \exists P.E \text{ occurs in } O$$

$$\mathbf{A}^-_{\neg}\ \frac{\neg D(c) \quad D(c)}{\bot(c)} \qquad \mathbf{A}^+_{\sqcup}\ \frac{D(c)}{D_1 \sqcup D_2(c)} : \begin{array}{l} D = D_1 \text{ or } D = D_2 \\ D_1 \sqcup D_2 \text{ occurs in } O \end{array}$$

$$\mathbf{A}^-_{\mathsf{inv}}\ \frac{P^-(c,d)}{P(d,c)} \qquad \mathbf{A}^+_{\mathsf{inv}}\ \frac{P(c,d)}{P^-(d,c)} : P^- \text{ occurs in } O$$

Fig. 1. Inference rules for ABox reasoning in $\mathcal{RL}$*tiny*

Web, but it is also typical for OBDA applications. Accordingly, many RDF databases also support some amount of OWL RL reasoning.

We have seen in Section 2.2 that the standard reasoning tasks of OWL can all be reduced to one another. Hence, in principle, it does not seem to matter for which of them we specify a reasoning method. However, the reduction of one reasoning task to another requires the ontology to be modified, which is not always convenient, especially if many queries should be answered at once (e.g., if many users of an ontology-based system submit queries). Moreover, we typically want to retrieve *all* instances of a certain class or property, not just check if some particular instance is contained. To do this efficiently, we need an approach that computes many instance relationships at once, rather than requiring us to check every possible instance individually.

A suitable calculus for instance retrieval is specified in Fig. 1. It consists of a list of *inference rules* of the form

$$\frac{\text{Premise}}{\text{Conclusion}} : \text{Side condition,}$$

which can be read as follows: if the *Premise* is given and the *Side condition* is satisfied, then the *Conclusion* should be derived. We suggest the reader to have a closer look at the rules in Fig. 1 now to verify that each of them expresses a plausible inference. For example, rule $\mathbf{A}_{\sqsubseteq}$ states that, if c is in class D, and D is a subclass of E, then c is also in E. The symbols c, D, and E should be thought of as place holders that can represent arbitrary expressions of the respective type. In other words, every inference rule in Fig. 1 is actually a template for many concrete *instances* of this inference rule. Unless we want to be very formal, we will usually ignore the difference between an inference rule and an instance of this inference rule. A rule is *applicable* to an ontology O if it has an instance such that O contains the respective premises and satisfies the respective side conditions. In this case, the rule can be *applied*, i.e., the conclusion of the (instance of the) rule can be added to O.

Let us look at a concrete example to see how we can apply the rules of Fig. 1 to derive new conclusions:

Example 22. Consider the TBox given in Example 20 together with two ABox axioms:

$$\mathit{FelisCatus}(\mathit{silvester}) \tag{21}$$
$$\mathit{preysOn}(\mathit{silvester}, \mathit{tweety}) \tag{22}$$

Applying the inference rules of Fig. 1, we can draw the following conclusions, where each line specifies the applied rule and premises (and the side condition in case of rule $\mathbf{A}_{\sqsubseteq}$):

$$\begin{array}{rrr}
\mathit{Animal}(\mathit{silvester}) & \mathbf{A}_{\sqsubseteq} : (21), (13) & (23) \\
\forall \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small})(\mathit{silvester}) & \mathbf{A}_{\sqsubseteq} : (21), (11) & (24) \\
\mathit{Animal} \sqcap \mathit{Small}(\mathit{tweety}) & \mathbf{A}^{-}_{\forall} : (24), (22) & (25) \\
\mathit{Animal}(\mathit{tweety}) & \mathbf{A}^{-}_{\sqcap} : (25) & (26) \\
\mathit{Small}(\mathit{tweety}) & \mathbf{A}^{-}_{\sqcap} : (25) & (27) \\
\exists \mathit{preysOn}.\mathit{Animal}(\mathit{silvester}) & \mathbf{A}^{+}_{\exists} : (22), (26) & (28) \\
\mathit{Animal} \sqcap \exists \mathit{preysOn}.\mathit{Animal}(\mathit{silvester}) & \mathbf{A}^{+}_{\sqcap} : (23), (28) & (29) \\
\mathit{Predator}(\mathit{silvester}) & \mathbf{A}_{\sqsubseteq} : (29), (12) & (30)
\end{array}$$

We have thus derived that Silvester is a predator.

The previous example illustrates the interplay of rules in our calculus. Class inclusions are taken into account by the single rule $\mathbf{A}_{\sqsubseteq}$. The remaining rules are used to derive facts that are immediate from the structure of class (and property) expressions. The $\mathbf{A}^{-}$ rules thereby decompose an axiom into smaller axioms, while the $\mathbf{A}^{+}$ rules assemble smaller parts into bigger expressions. The main goal of assembling bigger axioms is to make rule $\mathbf{A}_{\sqsubseteq}$ applicable, and this is why side conditions restrict $\mathbf{A}^{+}$ rules to produce only expressions that occur in $\mathcal{O}$. The example shows the typical sequence of derivations: in (24) a new complex axiom is derived using a class inclusion; (25) to (27) decompose this statement; (28) and (29) compose a new axiom; (30) applies another class inclusion to the result.

This also explains why we include rules $\mathbf{A}^{+}_{\exists}$ and $\mathbf{A}^{+}_{\sqcup}$ that derive $\mathcal{ALCI}$ ABox axioms that are not allowed in $\mathcal{RL}$*tiny*. Indeed, the task of the $\mathbf{A}^{+}$ rules is to derive the *left-hand side* of a class inclusion axiom, i.e., an axiom of the form **CL**(**IName**) rather than an $\mathcal{RL}$*tiny* ABox axiom **CR**(**IName**).

Finally, note that both premises and side conditions in our calculus play the role of preconditions that must hold in order to apply some rule. However, the application of a rule can only produce new premises, and will not affect the side conditions that hold. This is useful for implementing rule-based reasoning, where performance crucially depends on how efficiently one can check which rules are applicable.

Summary Rule-based reasoning methods apply a set of inference rules to derive new consequences. The rules in Fig. 1 produce consequences of $\mathcal{RL}$*tiny* ontologies.

3.3 Defining a Saturation Calculus for $\mathcal{RL}\mathit{tiny}$

The previous section introduced the concept of inference rules and gave an example where a valid consequence could be obtained. For a more systematic study of this reasoning approach, we need to be a bit more precise in defining the results of applying our calculus. This is the goal of the next definition:

Definition 23. An ontology is *saturated* under the rules of Fig. 1 if it contains the conclusions of all applicable rules. The *saturation* of an ontology $\mathcal{O}$ is the least saturated set $\mathcal{O}'$ that contains $\mathcal{O}$.

The calculus of Fig. 1 *derives* an axiom *ax* from $\mathcal{O}$ if one of the following holds:

– the saturation of $\mathcal{O}$ contains the axiom *ax*, or
– the saturation of $\mathcal{O}$ contains the axiom $\bot(c)$ for some individual c.

Most of this definition should simply spell out our intuitive idea of what it means to compute derivations based on a set of inference rules. In general, there are many saturated sets that contain an ontology; e.g., the set of all $\mathcal{ALCI}$ axioms over the given signature is always saturated. We thus refer to the *least* saturated superset of $\mathcal{O}$ to make sure that the derivations of the calculus correspond to the minimal amount of consequences that are really necessary. It is clear that any consequence that is derived by repeated application of rules must also be part of this least saturated ontology. The case where $\bot(c)$ is derived occurs if the ontology is inconsistent and thus entails every axiom, including *ax*.

Remark 24. This type of rule-based reasoning is known under various names, since it can be found in various forms in many areas of knowledge representation and artificial intelligence. Related names include:

– Saturation: we saturate an input; common term in theorem proving
– Deductive closure: another name for saturation; alludes to the notion of a closure operator
– Materialisation: derivations are "materialised," i.e., made explicit and stored; common term in the database literature
– Bottom-up reasoning: we start from the given ontology for computing derivations; commonly used in contrast to top-down reasoning that starts from a query (see Section 3.8)
– Forward chaining: rules are applied in a forward manner, from premise to conclusion; common term in rule-based knowledge representation and logic programming; the opposite is backward chaining, of which we see an example in Section 3.10
– Consequence-based reasoning: we produce consequences based on what is given, rather than trying to prove a hypothesis; term used in description logics, where many reasoning methods are guided by a hypothesis rather than by the given input

Definition 23 thus unambiguously specifies what our calculus derives. But is this really true? The definition is based on the saturation, the least saturated superset of $\mathcal{O}$. How can we be sure that such a set exists in all cases? So far, our intuition has been that the saturation is the "limit" of applying inference rules to $\mathcal{O}$, but rule application is not completely deterministic. One can often apply multiple rules at the same time, so there are many possible saturation processes for some ontologies. How can we be sure that the overall result does not depend on the order of rule applications? Indeed, it could be that there are multiple saturated sets, none of which is the least.

Example 25. To illustrate this problem, consider the following inference rules:

$$\frac{D_1 \sqcup D_2(c)}{D_1(c)} : D_2(c) \text{ was not derived} \qquad \frac{D_1 \sqcup D_2(c)}{D_2(c)} : D_1(c) \text{ was not derived}$$

Together, these rules ensure that every instance of a union $D_1 \sqcup D_2$ is also derived to be an instance of at least one of D_1 and D_2. This captures the semantics of $\sqcup$, but it prevents the existence of a unique saturation. Indeed, the ontology $O = \{D_1 \sqcup D_2(c)\}$ in contained in two saturated sets

$$\{D_1 \sqcup D_2(c), D_1(c)\} \quad \text{and} \quad \{D_1 \sqcup D_2(c), D_2(c)\}$$

but neither of these sets is the smaller than the other, and all sets that are contained in both are not saturated. Hence, the least saturated set that contains O does not exist.

Fortunately, the calculus in Fig. 1 has unique saturations. This is intuitively plausible since no rule application can prevent the application of any other rule. So whenever there are multiple possibilities for applying rules, it does not matter what we chose to do first: the other rule applications will still be possible later on. Formally, this property of the rule set is called *monotonicity*, meaning that a rule (instance) that is applicable to some ontology is also applicable to every larger ontology. This is enough to guarantee the existence of a unique saturation. A formal argument could look as follows:

Theorem 26. The saturation of an ontology under the rules in Fig. 1 exists. In particular, Definition 23 is well-defined.

Proof. We show the result by constructing the saturation as follows. Let $\hat{O}$ be the set intersection of all saturated ontologies that contain O. Note that the set of all axioms over the signature of O is always saturated, so $\hat{O}$ is an intersection of one or more sets. We show that $\hat{O}$ is the saturation that we are looking for.

Since $\hat{O}$ is an intersection of sets that contain O, we find $O \subseteq \hat{O}$. To establish the claim it remains to show that $\hat{O}$ is also saturated. To this end, we have to show that $\hat{O}$ contains the consequences of every rule instance that is applicable to $\hat{O}$. To do this, consider an arbitrary rule instance $\mathbf{R}$ that is applicable to $\hat{O}$. We must show that the consequence of $\mathbf{R}$ is in $\hat{O}$.

To this end, consider an arbitrary saturated ontology O' that contains O. Then $\hat{O} = \hat{O} \cap O'$ by our definition of $\hat{O}$. In other words, $\hat{O} \subseteq O'$. Since the inference rules are monotone, the rule instance $\mathbf{R}$ is applicable to O' as well. Since O' is saturated, we can conclude that O' contains the consequence of $\mathbf{R}$. Since O' was arbitrary, we have thus shown that every saturated ontology that contains O also contains the consequence of $\mathbf{R}$. Therefore, $\hat{O}$ contains the consequence of $\mathbf{R}$. Since $\mathbf{R}$ was arbitrary, this applies to all instances of all inference rules – in other words, $\hat{O}$ is saturated.

Summing up, we have shown that $\hat{O}$ is a saturated ontology that contains O, and that (by constriction) is contained in all other saturated ontologies. Therefore, $\hat{O}$ is the desired saturation. □

This proof works for all monotone inference systems, and is therefore often omitted as "obvious." However, even if the argument does not need to be repeated in every case, one should never forget to verify that the inference rules are indeed monotone. If a rule depends on the *absence* of some axiom, or if the application of a rule *deletes* an axiom rather than adding one, then it is no longer clear that a saturation exists. Many unwary computer scientists have fallen into that trap.

Summary The derivations of a calculus of monotone inference rules are defined by considering the saturation of an input ontology. The rules in Fig. 1 are monotone.

3.4 Correctness of the $\mathcal{RL}$*tiny* Instance Retrieval Calculus

Now that we have clearly defined a reasoning procedure based on the rules of Fig. 1, we would like to know if this calculus is correct (and in which sense). As discussed in Section 2.3, this question relates to soundness and completeness. To obtain a practical procedure, it should further be guaranteed that derivations can be computed in finite time. We thus arrive at the following three questions:

(1) Will a (naive) implementation of the calculus *terminate*, i.e., is the computation guaranteed to halt after a finite number of computation steps?
(2) Is the calculus *sound*, i.e., does it only produce valid consequences?
(3) Is the calculus *complete*, i.e., does it produce all valid consequences?

In this section, we will explain in some detail how one can prove termination and soundness. Completeness requires a bit more effort and is therefore discussed separately in Section 3.5. Proving these properties is not just an academic exercise but constitutes an important part of verifying the quality of any software system that is based on this calculus. When compared to other tasks of ensuring software quality, such as unit testing or profiling, the effort for establishing the correctness of the approach is rather small, but it requires a different set of techniques. The basic ideas found in this section and the next are used in many correctness proofs, but are rarely explained in full detail. Here, we give a more verbose and slow-paced presentation.

Termination We begin by showing termination, which will also have the important side effect of connecting our intuitive understanding of the calculus ("apply rules until nothing changes") to the formal definition in Section 3.3 ("consider the least saturated superset"). Indeed, we have only established the existence of the saturation by constructing it as the intersection of all saturated supersets, not by showing that the rules will lead to this result.

To obtain termination, we show that the repeated application of inference rules to O will stop after a finite number of steps. The simple reason for this is that there are only finitely many different axioms that the rules can derive from a given input. Indeed, every class expression, property expression, and individual in the conclusion of a rule must occur in its premises or in the input ontology. Thus, ultimately, all expressions in derivations occur in the input ontology. Since there are only finitely many such expressions in a finite input, the number of possible combinations of these expressions must also be finite. This is an induction argument, which can be made formal as follows:

Theorem 27. The repeated application of the rules of Fig. 1 to an ontology $\mathcal{O}$ terminates after at most $\mathsf{O}(s^3)$ steps, where s is the size of $\mathcal{O}$.[6]

Proof. We first show that every axiom *ax* that can be derived by a finite number n of rule applications contains only class expressions, property expressions, and individuals that also occur in the input ontology. This is easily done by induction over n.

If $n = 0$ (base case), then $ax \in \mathcal{O}$ and the claim is immediate. For the induction step, we assume that the claim has been shown for all axioms that are derived in less than n steps (induction hypothesis). The axiom *ax* must be obtained by applying some rule from Fig. 1 for which the premises have been derived before. By induction hypothesis, the premises only use expressions that occur in $\mathcal{O}$. By inspecting each individual rule, we find that *ax* also satisfies the claim. This finishes the induction.

Since every possible conclusion contains at most three expressions (classes, properties, individuals), this bounds the total number of derivations that can be obtained in a finite number of steps by $\mathsf{O}(s^3)$. This implies that the derivation terminates after at most that many steps. □

It is clear that the finite set of axioms that is obtained after termination is saturated. Moreover, it only contains axioms that are strictly necessary for this property (again, we could formalise this with an induction argument). Therefore, the set of derivations obtained by exhaustively applying the rules is indeed the saturation of an ontology in the sense of Definition 23.

Note that Theorem 27 does not state that the saturation can be computed in polynomial time. For this, we also need to observe that the applicability of a single rule can be checked in polynomial time. Since the total number of successful rule applications is linearly bounded by s^3, this leads to the overall polynomial bound.

Soundness Verifying that a set of inference rules is sound is often not a difficult task. In essence, we merely need to check that every rule will derive valid consequences provided that the inputs are valid. To make this formal, we need to refer to the formal semantics which defines the meaning of *valid*. Yet, we only need to show it under Direct Semantics: the case of RDF-Based Semantics then follows from Theorem 17. The following result illustrates how to formalise and prove a soundness result:

Proposition 28. Rule $\mathbf{A}_{\forall}^{-}$ is sound, i.e., for any ontology $\mathcal{O}$ with $\mathcal{O} \models \forall P.E(c)$ and $\mathcal{O} \models P(c, d)$, we have $\mathcal{O} \models E(d)$.

Proof. According to the definition of the Direct Semantics in Section 2.6, we need to show that every interpretation that satisfies $\mathcal{O}$ also satisfies $E(d)$. To this end, let $\mathcal{I}$ be an arbitrary interpretation with $\mathcal{I} \models \mathcal{O}$. Thus $\mathcal{I} \models P(c, d)$, that is, $\langle c^{\mathcal{I}}, d^{\mathcal{I}} \rangle \in P^{\mathcal{I}}$. Moreover, $\mathcal{I} \models \forall P.E(c)$, that is, $c^{\mathcal{I}} \in (\forall P.E)^{\mathcal{I}}$. According to Table 2, this means that for all f with $\langle c^{\mathcal{I}}, f \rangle \in P^{\mathcal{I}}$ we have $f \in E^{\mathcal{I}}$. Thus, in particular, $d^{\mathcal{I}} \in E^{\mathcal{I}}$, that is, $\mathcal{I} \models E(d)$. Since $\mathcal{I}$ was arbitrary, this finishes the proof. □

[6] It is inessential how exactly we define the size of $\mathcal{O}$. We could, e.g., take the length of $\mathcal{O}$ when written down as a set of axioms in DL syntax. However, the number of axioms in $\mathcal{O}$ (i.e., its cardinality) is not a good measure of size since it ignores the size of individual axioms.

The cases of the other rules are similar. The side conditions are only relevant for rule $\mathsf{A}_\sqsubseteq$. For all other rules, they are only used to avoid unnecessary derivations, which is important for termination but not for soundness.

Now given that the initial ontology $\mathcal{O}$ itself is certainly a consequence of $\mathcal{O}$, and that every rule can derive only valid consequences from valid premises, it is easy to see that only valid consequences will ever be derived.

Theorem 29. The calculus in Fig. 1 is sound, i.e., all axioms in the saturation of an ontology are logical consequences of this ontology.

Proof. Consider an axiom *ax* in the saturation of an ontology $\mathcal{O}$. Then *ax* can be obtained by applying inference rules to $\mathcal{O}$. Let n be the number of rule applications needed to derive *ax*. The fact that this number n is finite follows from our above discussion on termination. The induction proceeds as in the proof of Theorem 27, where we use the correctness of individual rules (e.g., Proposition 28) in the induction step. □

Again, this is a standard proof scheme that is rarely explicated in such detail. The essential ingredient is that every rule is sound in the sense of Proposition 28.

Summary The correctness of a calculus involves checking soundness, completeness, and termination. Termination and soundness can be shown by verifying that each rule preserves the required property (boundedness and validity of derivations, respectively).

3.5 Completeness of the $\mathcal{RL}$*tiny* Instance Retrieval Calculus

In this section, we explain how to prove that the instance retrieval calculus for $\mathcal{RL}$*tiny* is complete. This is often the most challenging task for proving the correctness of a reasoning procedure. As a first step, we need to understand what the expected output of the reasoning calculus is. Typically, we do not expect all valid results to be computed, but only valid results of a certain form. In our case, the calculus is expected to compute ABox axioms (since it is for instance retrieval). However, we already noted that the calculus does not compute arbitrary $\mathcal{RL}$*tiny* ABox axioms but rather axioms of the form **CL**(**IName**). Indeed, the following example illustrates that the calculus is not complete for $\mathcal{RL}$*tiny* ABox axioms:

Example 30. The following $\mathcal{RL}$*tiny* ontology is saturated under the rules of Fig. 1:

$$\{D(c), D \sqcap E \sqsubseteq \bot\}$$

This ontology entails the $\mathcal{RL}$*tiny* ABox axiom $\neg E(c)$ but this axiom is not in the saturation.

Moreover, the calculus can only compute axioms that are built from expressions that occur in the ontology: this is clear for A^- rules, and it is enforced by the side condition in A^+ rules. Summing up, we arrive at the following completeness statement that we would like to establish:

Theorem 31. The calculus in Fig. 1 is complete in the following sense:

– If $O \models E(c)$ where $E \in \mathbf{CL}$ and E occurs in O, then $E(c)$ is derived by the calculus.
– If $O \models P(c, d)$ where $P \in \mathbf{P}$ and P occurs in O, then $P(c, d)$ is derived by the calculus.

Note that *derived by the calculus* includes the two possible conditions of Definition 23. In spite of the restrictions, Theorem 31 still ensures that we can use the calculus to compute the instances of arbitrary class and property names. Class names are certainly in **CL**, and neither class nor property names require us to check if they really occur in O. If not, then they will never have any instances, so there is no danger of incompleteness in this case.

It remains to show that Theorem 31 is actually true. How can we possibly be sure that no important consequences are missed? The proof idea here (and in many other completeness proofs) is to show the *contrapositive*: whenever an axiom as in the claim is *not* derived by the calculus, the axiom is also *not* entailed by O. To show that an axiom is not entailed, one can provide a *counter-model*, i.e., an interpretation that satisfies O but that does not satisfy the respective axiom. If such an interpretation exists, then it is clear that not all interpretations that satisfy O satisfy the axiom – in other words, the axiom is not a logical consequence.

In the case of $\mathcal{RL}$*tiny* it is particularly easy to find a good counter-model. In fact, we can almost directly translate the saturated ontology into an interpretation, and this unique interpretation will work as a counter-model for all facts that have not been derived. We have already mentioned that OWL RL can be viewed as a kind of rule language, and indeed this construction is related to the *least Herbrand model* used in Logic Programming. To formalise this argument, we must mainly check that the axioms that are true in our chosen counter-model correspond to the axioms that are derived in the saturation. Doing this for class and property expressions of all relevant forms requires us to consider many cases, which makes the proof somewhat lengthy. Nevertheless, all of the individual steps should be easy to follow:

Proof (of Theorem 31). Consider an $\mathcal{RL}$*tiny* ontology O. Let O' be the saturation of O under the rules of Fig. 1. If O' contains an axiom of the form $\bot(c)$, then the claim follows immediately since in this case the calculus is defined to entail every axiom.

For the remainder, we assume that O' does not contain an axiom of the form $\bot(c)$. We define an interpretation $\mathcal{I}$ as follows:

– The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is the set of all individual symbols in the signature (without loss of generality, we can assume that there is at least one, even if it does not occur in O).
– For every individual symbol c, define $c^{\mathcal{I}} := c$.
– For every class name A, define $c \in A^{\mathcal{I}}$ if and only if $A(c) \in O'$.
– For every property name P, define $\langle c, d \rangle \in P^{\mathcal{I}}$ if and only if $P(c, d) \in O'$.

We want to show that $\mathcal{I}$ is a model of O that satisfies only axioms that either occur in O' or that are not of the forms defined in the claim of Theorem 31. We first observe that the relationship used to define $\mathcal{I}$ for property names extends to inverse properties:

Claim 1 The following statements are equivalent:

– P^- occurs in O and $\langle c, d\rangle \in P^{-\mathcal{I}}$
– $P^-(c, d) \in O'$

If the first statement holds, then $\langle d, c\rangle \in P^{\mathcal{I}}$ by the semantics of inverse properties. Thus $P(d, c) \in O'$ and rule $\mathsf{A}^+_{\mathsf{inv}}$ is applicable. Since O' is saturated, $P^-(c, d) \in O'$. Conversely, if the second statement holds, then $\mathsf{A}^-_{\mathsf{inv}}$ is applicable. Thus $P(d, c) \in O'$. By the definition of $\mathcal{I}$, $\langle d, c\rangle \in P^{\mathcal{I}}$. Therefore $\langle c, d\rangle \in P^{-\mathcal{I}}$, as required. The fact that P^- occurs in O follows since either $P^-(c, d) \in O'$ was derived by rule $\mathsf{A}^+_{\mathsf{inv}}$, or $P^-(c, d) \in O$.

This completes the proof of Claim 1. Lifting the definition of $\mathcal{I}$ to arbitrary **CL** classes requires a bit more effort. We first show the following direction:

Claim 2 If $E \in$ **CL** occurs in O, then $c \in E^{\mathcal{I}}$ implies $E(c) \in O'$.

In other words, all relationships $c \in E^{\mathcal{I}}$ that hold in $\mathcal{I}$ are also derived in O'. To show this claim for arbitrary **CL** classes E, we perform an induction over the structure of such classes, as defined by the grammar in Section 3.1. We begin with the base cases:

– If E is a class name, then the claim follows from the definition of $E^{\mathcal{I}}$.
– If $E = \bot$, then the claim holds since $c \in E^{\mathcal{I}}$ does not hold for any c (since $\bot^{\mathcal{I}} = \emptyset$).

For the remaining cases, we assume that the claim has already been established for the classes D, D_1 and D_2 (induction hypothesis). In all cases, we assume that E occurs in O and that $c \in E^{\mathcal{I}}$.

– Case $E = D_1 \sqcap D_2$. By the semantics of $\sqcap$, we find $c \in D_1^{\mathcal{I}}$ and $c \in D_2^{\mathcal{I}}$. Clearly, D_1 and D_2 occur in O since E does. Thus, the induction hypothesis implies $D_1(c) \in O'$ and $D_2(c) \in O'$. Since E occurs in O, rule $\mathsf{A}^+_{\sqcap}$ applies and $E(c) \in O'$ as required.
– Case $E = D_1 \sqcup D_2$. By the semantics of $\sqcup$, we find $c \in D_1^{\mathcal{I}}$ or $c \in D_2^{\mathcal{I}}$. Clearly, D_1 and D_2 occur in O since E does. Thus, the induction hypothesis implies $D_1(c) \in O'$ or $D_2(c) \in O'$. Since E occurs in O, rule $\mathsf{A}^+_{\sqcup}$ applies and $E(c) \in O'$ as required.
– Case $E = \exists P.D$. By the semantics of $\exists$, there is an element $d \in \Delta^{\mathcal{I}}$ such that $\langle c, d\rangle \in P^{\mathcal{I}}$ and $d \in D^{\mathcal{I}}$. By the definition of $\mathcal{I}$, d is an individual name. Since E occurs in O, so do P and D. According to Claim 1 (if P is inverse) or the definition of $\mathcal{I}$ (otherwise), we find that $P(c, d) \in O'$. By the induction hypothesis, $D(d) \in O'$. Since E occurs in O, rule $\mathsf{A}^+_{\exists}$ applies and $E(c) \in O'$ as required.

This finishes the proof of Claim 2. The other direction needs to be established for another language of class expressions:

Claim 3 If $E \in$ **CR** and $E(c) \in O'$, then E occurs in O and $c \in E^{\mathcal{I}}$.

The fact that E occurs in O is not hard to see. Given only premises with classes in O, every rule will only introduce classes with that property (in the A^+ rules this is ensured by side conditions). So this part of the claim can be shown with an easy induction, similar to the one we did for Theorem 27.

To show the remaining claim for arbitrary **CR** classes E, we perform an induction over the structure of such classes, as defined by the grammar in Section 3.1. We begin with the base cases:

- If E is a class name, then the claim follows from the definition of $E^{\mathcal{I}}$.
- If $E = \bot$, then the claim holds since we assumed that no axiom of the form $E(c)$ occurs in O'.

For the remaining cases, we assume that the claim has already been established for the classes D, D_1 and D_2 (induction hypothesis). In all cases, we assume that $E(c) \in O'$.

- Case $E = D_1 \sqcap D_2$. Then rule $\mathbf{A}_{\sqcap}^{-}$ is applicable. Since O' is saturated, we find $D_1(c) \in O'$ and $D_2(c) \in O'$. By the induction hypothesis, $c \in D_1^{\mathcal{I}}$ and $c \in D_2^{\mathcal{I}}$. By the semantics of $\sqcap$, this implies $c \in E^{\mathcal{I}}$ as required.
- Case $E = \neg D$ where $D \in \mathbf{CL}$. We have assumed that O' does not contain the axiom $\bot(c)$. Therefore, rule $\mathbf{A}_{\neg}^{-}$ is not applicable, that is, $D(c) \notin O'$. We already noted above that E and thus D must occur in O. Therefore, (the contrapositive of) Claim 2 implies that $c \notin D^{\mathcal{I}}$. By the semantics of $\neg$, $c \in E^{\mathcal{I}}$.
- Case $E = \forall P.D$. Consider an arbitrary element d such that $\langle c, d\rangle \in P^{\mathcal{I}}$. According to Claim 1 (if P is inverse) or the definition of $\mathcal{I}$ (otherwise), we find that $P(c, d) \in O'$. Therefore, rule $\mathbf{A}_{\forall}^{-}$ is applicable to obtain $D(d) \in O'$. By the induction hypothesis, this implies $d \in D^{\mathcal{I}}$. Since d was arbitrary, this shows that $c \in E^{\mathcal{I}}$ according to the semantics of $\forall$.

This finishes the proof of Claim 3. We can now show that $\mathcal{I}$ is a model of O. We need to show that $\mathcal{I}$ satisfies each axiom in O. We distinguish the possible forms of $\mathcal{RL}$*tiny* axioms:

- ABox axioms $E(c) \in O$: then $E \in \mathbf{CR}$ occurs in O, so $c \in E^{\mathcal{I}}$ follows from Claim 3.
- ABox axioms $P(c, d) \in O$: then $\langle c, d\rangle \in P^{\mathcal{I}}$ follows from Claim 1 (if P is inverse) or the definition of $\mathcal{I}$ (otherwise).
- TBox axioms $D \sqsubseteq E \in O$. Whenever there is an element $c \in D^{\mathcal{I}}$, then $D(c) \in O'$ by Claim 2. Thus, rule $\mathbf{A}_{\sqsubseteq}$ is applicable and yields $E(c) \in O'$. By Claim 3, $c \in E^{\mathcal{I}}$.

This shows that $\mathcal{I}$ models all axioms of O. Finally, the two cases of the overall claim can be shown by a similar argument:

- If $E(c) \notin O'$ for some $E \in \mathbf{CL}$ that occurs in O, then $c \notin E^{\mathcal{I}}$ by (the contrapositive of) Claim 2. Hence $O \not\models E(c)$.
- If $P(c, d) \notin O'$ for some P that occurs in O, then $\langle c, d\rangle \notin P^{\mathcal{I}}$ by (the contrapositive of) Claim 1 and the definition of $\mathcal{I}$. Hence $O \not\models P(c, d)$.

We have thus shown that, whenever axioms of this type are not derived, they are not logical consequences of O. This finishes the claim. □

We have thus convinced us that our calculus is indeed complete. In long proofs, it is easy to overlook some possible cases. A good cross check for the completeness of our completeness proof is therefore that every rule in the calculus is actually mentioned somewhere. If we had been able to show completeness without using each rule, then we would either have forgotten something, or the calculus would contain more rules than needed for deriving all results (this is usually bad in practice, since it means that algorithms have more possibilities for deriving the same result redundantly).

$$\mathbf{T}_{\sqsubseteq}\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in O \qquad\qquad \mathbf{T}^{+}_{\mathbf{i}}\ \frac{}{C \sqsubseteq C \quad C \sqsubseteq \top} : C \text{ occurs in } O$$

$$\mathbf{T}^{-}_{\sqcap}\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad\qquad \mathbf{T}^{+}_{\sqcap}\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occurs in } O$$

$$\mathbf{T}^{-}_{\exists}\ \frac{C \sqsubseteq \exists P.\bot}{C \sqsubseteq \bot} \qquad\qquad \mathbf{T}^{+}_{\exists}\ \frac{C \sqsubseteq \exists P.D \quad D \sqsubseteq E}{C \sqsubseteq \exists P.E} : \exists P.E \text{ occurs in } O$$

Fig. 2. Inference rules for TBox reasoning in $\mathcal{EL}$*tiny*

Summary Completeness of reasoning algorithms can often been shown by constructing counter-models to show that axioms that are not derived are not true in all models either.

3.6 A Rule-based Classification Calculus for $\mathcal{EL}$*tiny*

We now consider the reasoning task of classification in OWL EL (or rather $\mathcal{EL}$*tiny*) ontologies. It turns out that a suitable reasoning method is actually very similar to the instance retrieval calculus that we have considered for $\mathcal{RL}$*tiny*, in spite of the difference in ontology language and reasoning task.

Classification is the most important inference task for OWL EL. Many applications of EL involve ontologies that are built by experts and that can be used in many concrete scenarios. For example, the ontology *SNOMED CT* (Clinical Terms) defines about 300,000 classes about human diseases and related concepts. It does not contain any individuals, since individual patients and diseases are only added when SNOMED CT is deployed, e.g., in hospitals. In the case of SNOMED CT, the class hierarchy is even precomputed before shipping the ontology to users. OWL is therefore used to simplify the development of the ontology: instead of modelling a huge taxonomy of medical terms, experts describe terms in OWL so that the taxonomy can be computed automatically.

As in the case of instance retrieval, classification could be done by checking all possible class subsumptions individually, but we are interested in a more efficient algorithm that can compute all valid subsumptions in a single run. A suitable set of inference rules is given in Fig. 2. We are already familiar with this kind of inference rules from the $\mathcal{RL}$*tiny* instance retrieval calculus. The main difference is that we are now computing class subsumptions rather than ABox axioms. In fact, the rules in Fig. 2 do not take ABox axioms into account at all, i.e., they are tailored for ontologies that consist only of TBox axioms.

A closer look reveals that the inference rules have many similarities with those in Fig. 1. For example, if we compare rule $\mathbf{A}_{\sqsubseteq}$ with rule $\mathbf{T}_{\sqsubseteq}$, we see that the basic structure of the rule is very similar: we mainly replaced the individual c by the class C to change from ABox to TBox axioms. Likewise, the rules $\mathbf{A}^{-}_{\sqcap}$, $\mathbf{A}^{+}_{\sqcap}$, and $\mathbf{A}^{+}_{\exists}$ are similar to $\mathbf{T}^{-}_{\sqcap}$, $\mathbf{T}^{+}_{\sqcap}$, and $\mathbf{T}^{+}_{\exists}$, respectively. The main differences stem from the fact that many $\mathcal{RL}$*tiny* features are not relevant for $\mathcal{EL}$*tiny*. Moreover, we have an additional *initialisation rule* $\mathbf{T}^{+}_{\mathbf{i}}$ that requires no premises but produces multiple conclusions, and an additional rule $\mathbf{T}^{-}_{\exists}$ that propagates the empty class back along existential restrictions. The latter was not necessary in $\mathcal{RL}\mathit{tiny}$, where all existential quantifiers in derivations are derived by rule $\mathbf{A}^{+}_{\exists}$, such that the empty class would lead to an inconsistency there already. The following example illustrates the use of the calculus:

Example 32. Consider the TBox given in Example 19 which we repeat here for convenience:

$$\mathit{FelisCatus} \sqsubseteq \exists \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small}) \tag{31}$$
$$\mathit{Animal} \sqcap \exists \mathit{preysOn}.\mathit{Animal} \sqsubseteq \mathit{Predator} \tag{32}$$
$$\mathit{FelisCatus} \sqsubseteq \mathit{Animal} \tag{33}$$

Applying the inference rules of Fig. 2, we can draw the following conclusions, where each line specifies the applied rule and premises:

$$\begin{array}{rr}
\mathit{FelisCatus} \sqsubseteq \mathit{FelisCatus} & \mathbf{T}^{+}_{\mathbf{i}} \\
\mathit{FelisCatus} \sqsubseteq \top & \mathbf{T}^{+}_{\mathbf{i}} \\
\mathit{FelisCatus} \sqsubseteq \mathit{Animal} & \mathbf{T}_{\sqsubseteq} : (34), (33) \\
\mathit{FelisCatus} \sqsubseteq \exists \mathit{preysOn}.\mathit{Animal} \sqcap \mathit{Small} & \mathbf{T}_{\sqsubseteq} : (34), (31) \\
\mathit{Animal} \sqcap \mathit{Small} \sqsubseteq \mathit{Animal} \sqcap \mathit{Small} & \mathbf{T}^{+}_{\mathbf{i}} \\
\mathit{Animal} \sqcap \mathit{Small} \sqsubseteq \top & \mathbf{T}^{+}_{\mathbf{i}} \\
\mathit{Animal} \sqcap \mathit{Small} \sqsubseteq \mathit{Animal} & \mathbf{T}^{-}_{\sqcap} : (38) \\
\mathit{Animal} \sqcap \mathit{Small} \sqsubseteq \mathit{Small} & \mathbf{T}^{-}_{\sqcap} : (38) \\
\mathit{FelisCatus} \sqsubseteq \exists \mathit{preysOn}.\mathit{Animal} & \mathbf{T}^{+}_{\exists} : (37), (40) \\
\mathit{FelisCatus} \sqsubseteq \mathit{Animal} \sqcap \exists \mathit{preysOn}.\mathit{Animal} & \mathbf{T}^{+}_{\sqcap} : (36), (42) \\
\mathit{FelisCatus} \sqsubseteq \mathit{Predator} & \mathbf{T}_{\sqsubseteq} : (43), (32)
\end{array}
\begin{array}{r}
(34)\\(35)\\(36)\\(37)\\(38)\\(39)\\(40)\\(41)\\(42)\\(43)\\(44)
\end{array}$$

We have thus derived that all cats are predators.

The dynamics of this derivation are again similar to the case of $\mathcal{RL}\mathit{tiny}$: rules $\mathbf{T}_{\sqsubseteq}$ and $\mathbf{T}^{+}_{\mathbf{i}}$ are used to derive new class expressions, $\mathbf{T}^{-}$ rules decompose these expressions, and $\mathbf{T}^{+}$ rules are used to build more complex expressions that may be relevant for applying $\mathbf{T}_{\sqsubseteq}$ again. Note that we have used $\mathbf{T}^{+}_{\mathbf{i}}$ rather selectively in our example: in (34) to initialise the class *FelisCatus* we were interested in, and in (38) to initialise the expression $\mathit{Animal} \sqcap \mathit{Small}$ that we had just encountered as a filler for the existential in (37). These are indeed the only cases where $\mathbf{T}^{+}_{\mathbf{i}}$ needs to be used, and a more optimised calculus should restrict rule applications further to avoid unnecessary computations.

The application of inference rules for obtaining a saturation is defined as in the case of $\mathcal{RL}\mathit{tiny}$. It is easy to see that we can apply similar arguments as before to show that the unique saturation of every ontology exists and can be obtained in a finite number of steps by applying the rules. An axiom $C \sqsubseteq D$ is *derived* from an ontology O by the resulting calculus if any of the following axioms occurs in the saturation of O:

- $C \sqsubseteq D$ (axiom derived directly)
- $C \sqsubseteq \bot$ (class C is inconsistent)
- $\top \sqsubseteq \bot$ (ontology O is inconsistent)

Note that, in the absence of ABox axioms, there is only one possible form of inconsistency that can be derived. However, we need to take into account the case where $C \sqsubseteq \bot$: in this case, C must be empty and is therefore a subclass of any other class, but our calculus would not explicitly derive this.

Remark 33. In this chapter, we focus on $\mathcal{EL}\,tiny$ ontologies that contain subclass inclusion axioms only, since we want to illustrate the new ideas related to classification. However, it is not difficult to extend the rules of Fig. 2 to support instance retrieval for $\mathcal{EL}\,tiny$. Some of the required rules can just be adapted from Fig. 1. All rules in Fig. 1 are sound, i.e., they cannot lead to wrong results when applied to any ontology, but only rules $\mathsf{A}_\sqsubseteq$, $\mathsf{A}^-_\sqcap$, $\mathsf{A}^+_\sqcap$, and $\mathsf{A}^+_\exists$ are relevant for $\mathcal{EL}\,tiny$. The following additional rules are needed:

$$\mathsf{A}^+_\mathsf{i}\ \frac{}{\top(c)} : c \text{ occurs in } O \qquad \mathsf{A}^-_\exists\ \frac{\exists P.\bot(c)}{\bot(c)} \qquad \mathsf{AT}^+_\exists\ \frac{\exists P.D(c) \quad D \sqsubseteq E}{\exists P.E(c)}$$

Rules rules A^+_i and $\mathsf{A}^-_\exists$ are essentially the ABox versions of rules T^+_i and $\mathsf{T}^-_\exists$, respectively. Rule $\mathsf{AT}^+_\exists$ can be considered as a middle ground between rule $\mathsf{A}^+_\exists$ and $\mathsf{T}^+_\exists$.

These rules, including all of the rules in Fig. 2, together yield a complete instance retrieval calculus for $\mathcal{EL}\,tiny$. As in the case of $\mathcal{RL}\,tiny$, it is also necessary to check if derivations of the form $\bot(c)$ indicate that the ontology is inconsistent. Note that, even if we are only interested in deriving ABox axioms, we have to compute certain class subsumptions that might be needed as a second premise for rule $\mathsf{AT}^+_\exists$.

Summary Subclass inclusion axioms entailed by $\mathcal{EL}\,tiny$ ontologies can be computed with a rule-based saturation calculus, similar to the one for instance retrieval in $\mathcal{RL}\,tiny$.

3.7 Correctness of Rule-based Classification in $\mathcal{EL}\,tiny$

We still need to check that the classification calculus for $\mathcal{EL}\,tiny$ is really correct. In particular, we need to clarify for what kind of computations the calculus is complete. As in the case of $\mathcal{RL}\,tiny$, only certain kinds of logical consequences are computed, and we can only use the calculus in implementations if we properly understand what we can expect from it. The other aspects of correctness – soundness and termination – can be shown just as in the case of $\mathcal{RL}\,tiny$, so we will not discuss them in such detail again.

Concretely, termination follows since rules only derive axioms with subclass and superclass expressions that occur in the ontology. Therefore, there is at most a quadratic number of possible derivations. Note that this is even a lower estimate than for $\mathcal{RL}\,tiny$ (Theorem 27), where we could have a cubic number of derivations (by combining all properties with all possible pairs of individuals). Soundness follows by observing that each individual rule is sound (i.e., preserves the validity of its inputs).

To show completeness, we can also use some of the ideas that we introduced in Section 3.5 for $\mathcal{RL}\,tiny$. Namely, we take the approach of showing that all axioms (of a certain kind) that are not derived are no logical consequences either. To do this, we again construct a single counter-model that refutes all underived axioms.

The new challenge for $\mathcal{EL}\,tiny$ is that we cannot define a counter-model by simply taking individuals as domain elements and ABox facts to define the interpretation of classes and properties: there are no ABox facts in our case; there are not even individuals. So where should we get sufficiently many elements from to construct a counter-model? The solution is to use one element for every class name. The idea is that this element represents the properties that all representatives of this class must share. For example, the element $e_{\mathit{FelisCatus}}$ would represent the "typical cat" in our model. The derived class inclusion axioms are enough to know the relationship that such representative elements need to be in. For example, the axiom $\mathit{FelisCatus} \sqsubseteq \exists \mathit{preysOn}.(\mathit{Animal} \sqcap \mathit{Small})$ tells us that the "typical cat" preys on the "typical small animal." Furthermore, we need to take care to introduce representative elements only for classes that are not empty. If $C \sqsubseteq \bot$ was derived, then we should obviously not create an element for C.

The key insight here is that a single typical representative per class is enough to find a suitable counter-model. As an interesting side effect, this also ensures that the model is rather small, just as in the case of $\mathcal{RL}$*tiny*. This is not the case for OWL in general, where some satisfiable ontologies have only models that are very large or even infinite. Let us now formulate the exact kind of completeness that we want to show and see how exactly the counter-model can be constructed:

Theorem 34. The calculus in Fig. 2 is complete in the following sense: Consider an $\mathcal{EL}$*tiny* ontology O that consists only of class inclusion axioms. If $O \models C \sqsubseteq D$ where C and D occur in O, then $C \sqsubseteq D$ is derived by the calculus.

Proof. Consider an $\mathcal{EL}$*tiny* ontology O that contains no ABox axioms. Let O' be the saturation of O under the rules of Fig. 2. If O' contains the axiom $\top \sqsubseteq \bot$, then the claim follows immediately since in this case the calculus is defined to entail every axiom.

For the remainder, we assume that O' does not contain the axiom $\top \sqsubseteq \bot$. Moreover, for technical reasons that will become clear soon, we first show the claim only for ontologies O that contain $\top$. It will be easy to lift this restriction later. Given such an ontology O, we define an interpretation $\mathcal{I}$ as follows:

- The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is the set of all elements of the form e_C where C is a class expression that occurs in O and $C \sqsubseteq \bot \notin O'$.
- For every class name A, define $e_C \in A^{\mathcal{I}}$ if and only if $C \sqsubseteq A \in O'$.
- For every property name P, define $\langle e_C, e_D \rangle \in P^{\mathcal{I}}$ if and only if $C \sqsubseteq \exists P.D \in O'$.

Note that there are no individual symbols to be interpreted. The domain of $\mathcal{I}$ is not empty since O contains $\top$ and $\top \sqsubseteq \bot$ was not derived (this is why we assume that $\top$ is in O – otherwise we would have to introduce an additional element that would require separate consideration in all arguments). We want to show that $\mathcal{I}$ is a model of O that satisfies only axioms that either occur in O' or that are not of the form defined in the claim of Theorem 34.

Claim 1 If E occurs in O, then $e_C \in E^{\mathcal{I}}$ implies $C \sqsubseteq E \in O'$.

Note that this implies that C occurs in O; otherwise there would not be an element $e_C \in \Delta^{\mathcal{I}}$. To show this claim for arbitrary $\mathcal{EL}$*tiny* classes E, we perform an induction over the structure of such classes, as defined by the grammar in Section 3.1. We begin with the base cases:

- If E is a class name, then the claim follows from the definition of $E^{\mathcal{I}}$.

– If $E = \top$, then the claim holds since $C \sqsubseteq E \in \mathcal{O}'$ holds by rule $\mathbf{T}^+_{\mathsf{i}}$.
– If $E = \bot$, then the claim holds since $e_C \in E^\mathcal{I}$ does not hold for any e_C (since $\bot^\mathcal{I} = \emptyset$).

For the remaining cases, we assume that the claim has already been established for the classes D, D_1 and D_2 (induction hypothesis). In all cases, we assume that E occurs in $\mathcal{O}$ and that $e_C \in E^\mathcal{I}$.

– Case $E = D_1 \sqcap D_2$. By the semantics of $\sqcap$, we find $e_C \in D_1^\mathcal{I}$ and $e_C \in D_2^\mathcal{I}$. Clearly, D_1 and D_2 occur in $\mathcal{O}$ since E does. Thus, the induction hypothesis implies $C \sqsubseteq D_1 \in \mathcal{O}'$ and $C \sqsubseteq D_2 \in \mathcal{O}'$. Since E occurs in $\mathcal{O}$, rule $\mathbf{T}^+_\sqcap$ applies and $C \sqsubseteq E \in \mathcal{O}'$ as required.
– Case $E = \exists P.D$. By the semantics of $\exists$, there is an element $e_F \in \Delta^\mathcal{I}$ such that $\langle e_C, e_F \rangle \in P^\mathcal{I}$ and $e_F \in D^\mathcal{I}$. By the definition of $\mathcal{I}$, F must occur in $\mathcal{O}$, and $C \sqsubseteq \exists P.F \in \mathcal{O}'$. Since E occurs in $\mathcal{O}$, so does D. By the induction hypothesis, $F \sqsubseteq D \in \mathcal{O}'$. Since E occurs in $\mathcal{O}$, rule $\mathbf{T}^+_\exists$ applies and $C \sqsubseteq E \in \mathcal{O}'$ as required.

This finishes the proof of Claim 1. It remains to show the converse:

Claim 2 If $C \sqsubseteq E \in \mathcal{O}'$ and $e_C \in \Delta^\mathcal{I}$, then E occurs in $\mathcal{O}$ and $e_C \in E^\mathcal{I}$.

The fact that E occurs in $\mathcal{O}$ is not hard to see; we already noted this when discussing termination above. To show the remaining claim for arbitrary $\mathcal{EL}$*tiny* classes E, we perform another induction over the structure of such classes. We begin with the base cases:

– If E is a class name, then the claim follows from the definition of $E^\mathcal{I}$.
– If $E = \top$, then the claim holds since $e_C \in \Delta^\mathcal{I} = \top^\mathcal{I}$.
– If $E = \bot$, then the claim holds since $C \sqsubseteq E \notin \mathcal{O}'$, which follows from $e_C \in \Delta^\mathcal{I}$.

For the remaining cases, we assume that the claim has already been established for the classes D, D_1 and D_2 (induction hypothesis). In all cases, we assume that $C \sqsubseteq E \in \mathcal{O}'$ and $e_C \in \Delta^\mathcal{I}$.

– Case $E = D_1 \sqcap D_2$. Then rule $\mathbf{T}^-_\sqcap$ is applicable. Since $\mathcal{O}'$ is saturated, we find $C \sqsubseteq D_1 \in \mathcal{O}'$ and $C \sqsubseteq D_2 \in \mathcal{O}'$. By the induction hypothesis, $e_C \in D_1^\mathcal{I}$ and $e_C \in D_2^\mathcal{I}$. By the semantics of $\sqcap$, this implies $e_C \in E^\mathcal{I}$ as required.
– Case $E = \exists P.D$. We first show that $e_D \in \Delta^\mathcal{I}$. Suppose for a contradiction that $D \sqsubseteq \bot \in \mathcal{O}'$. Then $C \sqsubseteq \exists P.\bot \in \mathcal{O}'$ by rule $\mathbf{T}^+_\exists$. But then $C \sqsubseteq \bot \in \mathcal{O}'$ by rule $\mathbf{T}^-_\exists$, which contradicts our assumption that $e_C \in \Delta^\mathcal{I}$. Therefore $D \sqsubseteq \bot \notin \mathcal{O}'$.
Thus $e_D \in \Delta^\mathcal{I}$. By definition of $\mathcal{I}$, $\langle e_C, e_D \rangle \in P^\mathcal{I}$. By rule $\mathbf{T}^+_{\mathsf{i}}$, $D \sqsubseteq D \in \mathcal{O}'$, and therefore $e_D \in D^\mathcal{I}$. By the semantics of $\exists$, this implies $e_C \in E^\mathcal{I}$ as required.

This finishes the proof of Claim 2. We can now show that $\mathcal{I}$ is a model of $\mathcal{O}$. We need to show that $\mathcal{I}$ satisfies each axiom in $\mathcal{O}$. Consider an axiom $D \sqsubseteq E \in \mathcal{O}$. Whenever there is an element $e_C \in D^\mathcal{I}$, then $C \sqsubseteq D \in \mathcal{O}'$ by Claim 1. Thus, rule $\mathbf{T}_\sqsubseteq$ is applicable and yields $C \sqsubseteq E \in \mathcal{O}'$. By Claim 2, $e_C \in E^\mathcal{I}$.

This shows that $\mathcal{I}$ models all axioms of $\mathcal{O}$. To show the overall claim, consider a subclass inclusion $C \sqsubseteq D$ where C and D occur in $\mathcal{O}$, such that $C \sqsubseteq D$ is not derived.

In particular, this means that $C \sqsubseteq \bot \notin O'$. Thus there is an element $e_C \in \Delta^{\mathcal{I}}$. Since $C \sqsubseteq C \in O'$ by rule $\mathbf{T}_{\mathbf{i}}^{+}$, we also have $e_C \in C^{\mathcal{I}}$. However, by (the contrapositive of) Claim 1, $e_C \notin D^{\mathcal{I}}$. Therefore, $\mathcal{I} \not\models C \sqsubseteq D$. Summing up, whenever an axiom $C \sqsubseteq D$ is not derived, it is not a logical consequence of O.

This finishes the proof for all ontologies O that contain $\top$. If O does not contain $\top$, then we can construct the model $\mathcal{I}$ based on the extended ontology $O_{\top} := O \cup \{\top \sqsubseteq \top\}$. If O' is the saturation of O, then the saturation of $O_{\top}$ is $O' \cup \{\top \sqsubseteq \top\}$. This is easy to see: $O_{\top}$ may allow additional rule applications of rules $\mathbf{T}_{\sqsubseteq}$ and $\mathbf{T}_{\mathbf{i}}^{+}$, but the possible derivations are already contained in $O' \cup \{\top \sqsubseteq \top\}$. Therefore, the completeness result for $O_{\top}$ (established by our above proof) carries over to O. □

The counter-model that we have constructed in the above proof is also known as a *universal* or *canonical model* of the $\mathcal{EL}$*tiny* ontology. This name hints at the fact that all logical entailments and non-entailments are captured in this single model. It therefore generalises the idea of a least model that is used for rule languages (and also for $\mathcal{RL}$*tiny*). Not all OWL ontologies have a canonical model (e.g., in every model of the axiom $C \sqcup D(a)$, the individual a corresponds to an instance of C or D, but neither $C(a)$ nor $D(a)$ is a logical consequence; hence no model can be canonical).

Summary Completeness of the $\mathcal{EL}$*tiny* classification calculus is shown by constructing a canonical counter-model, which refutes all TBox axioms that have not been derived.

3.8 Query Rewriting for Reasoning in $\mathcal{QL}$*tiny*

We now turn our attention to reasoning in $\mathcal{QL}$*tiny*, where we will again focus on instance retrieval and a related (more difficult) task of *conjunctive query answering*. The reasoning method used for $\mathcal{QL}$*tiny* is rather different from the saturation-based approaches used for $\mathcal{RL}$*tiny* and $\mathcal{EL}$*tiny*. Instead of deriving all consequences (of a certain form) from an ontology, reasoning in $\mathcal{QL}$*tiny* is done by *query rewriting*. This works as follows:

(1) The user specifies a query, e.g., the query $\mathit{Animal}(x)$ to retrieve all animals.
(2) Using the TBox of the ontology, this query is rewritten into a set of queries. For example, if the ontology only states $\mathit{FelisCatus} \sqsubseteq \mathit{Animal}$, then the query would rewrite to two queries, $\mathit{Animal}(x)$ and $\mathit{FelisCatus}(x)$.
(3) The rewritten queries are answered using the ABox of the ontology only, i.e., they are matched to the facts. In our example, the answers to the queries would thus be all individuals a for which there is a fact $\mathit{Animal}(a)$ or a fact $\mathit{FelisCatus}(a)$ in the ontology.

It is guaranteed that the answers to the ABox queries of step (3) are exactly the answers to the original query (over ABox and TBox) of step (1). The advantage of this approach is that the TBox of the ontology is only needed in step (2), while step (3) can be solved by querying the ABox like a standard database. Indeed, the queries of step (3) can also be expressed in standard database query languages like SQL, and existing database management systems can be used to compute the query answers efficiently. The special feature of $\mathcal{QL}\,tiny$ is that the set of queries that is relevant in step (3) is finite. Example 43 below will illustrate that this is not the case for $\mathcal{EL}\,tiny$ and $\mathcal{RL}\,tiny$.

Before explaining $\mathcal{QL}\,tiny$ query rewriting in detail, we should clarify what we mean by *query*.[7] As in the example above, a query could simply be an ABox axiom where a variable is used instead of an individual name, e.g., $\mathsf{Animal}(x)$ or $\mathsf{preysOn}(x, y)$.[8] This provides a convenient syntax for specifying an instance retrieval problem. However, a more powerful query language is obtained by combining many such statements conjunctively. For example,

$$\mathsf{FelisCatus}(x) \wedge \mathsf{preysOn}(x, y) \wedge \mathsf{Animal}(y) \tag{45}$$

asks for all cats x and animals y such that x preys on y. This query consists of three expressions that are combined with conjunctions; each such expression in a query is called a *query atom*. The following example illustrates a subtle issue that we have to take into account when answering such queries over OWL ontologies.

Example 35. Consider the following $\mathcal{QL}\,tiny$ ontology:

$\mathsf{FelisCatus}(\mathsf{silvester})$	Silvester is a cat.	(46)
$\mathsf{FelisCatus}(\mathsf{tom})$	Tom is a cat.	(47)
$\mathsf{SerinusCanaria}(\mathsf{tweety})$	Tweety is a canary bird.	(48)
$\mathsf{preysOn}(\mathsf{silvester}, \mathsf{tweety})$	Silvester preys on Tweety.	(49)
$\mathsf{SerinusCanaria} \sqsubseteq \mathsf{Animal}$	All canary birds are animals.	(50)
$\mathsf{FelisCatus} \sqsubseteq \exists \mathsf{preysOn}.\mathsf{Animal}$	All cats prey on some animal.	(51)

Then the query (45) has the solution $x = \mathsf{silvester}$, $y = \mathsf{tweety}$. Indeed, (46) and (49) immediately show that the first two query atoms in the query are satisfied by this solution. The third atom is also satisfied, since $\mathsf{Animal}(\mathsf{tweety})$ is a consequence of (48) and (50).

In contrast, the query (45) has no solution with $x = \mathsf{tom}$. This might be surprising, since we know that Tom is a cat (47) and that all cats prey on some animal (51). However, we do not know of any concrete animal that Tom preys on, so we cannot find an assignment for y to construct a query solution.

The previous example shows that it makes a difference whether we know that an individual has certain properties, or whether we merely know that some element with these properties must exist. In the latter case, the *anonymous* element cannot be the answer to a query. It is therefore useful to have a way of specifying that a variable in a query should not be part of the answer, so that it is enough if some suitable value is known to exist. To express this syntactically, we bind such variables with existential quantifiers. For example,

$$\exists y.\mathsf{FelisCatus}(x) \wedge \mathsf{preysOn}(x, y) \wedge \mathsf{Animal}(y) \tag{52}$$

[7] For a more detailed (and more formal) introduction to conjunctive queries in DLs, see the chapter *Reasoning and Query Answering in Description Logics* in these lecture notes [33].

[8] We will always use x, y, z for variables, so that no confusion with individual names is likely.

specifies a query for all cats x that prey on some animal. The animal y is not part of the answer, and no concrete value has to be found for it. Thus, both $x = \mathsf{silvester}$ and $x = \mathsf{tom}$ would be answers to query (52) over the ontology in Example 35.

This completes the notion of conjunctive query, as summed up in the next definition:

Definition 36. A *conjunctive query Q* is a formula of the form $\exists \boldsymbol{y}.C_1 \wedge \ldots \wedge C_\ell$ where $\boldsymbol{y}$ is a list of variables and each *query atom C_i* is of one of the following forms:

– $A(x)$ where A is a class name, or
– $P(x, y)$ where P is a property name.

The variables $\boldsymbol{y}$ are the *non-distinguished variables* of Q. All other variables in Q are *distinguished*.

The semantics of conjunctive queries can be formalised by reading a query Q as a first-order logic formula. A solution mapping for a query is given by assigning an individual name to each distinguished variable. We call a solution mapping a solution (or an answer) to the query over some ontology, if the formula that we get by replacing each distinguished variable by its assigned individual name is a logical consequence of the ontology (under first-order logic semantics). For most of this section, it suffices to have an intuitive understanding of the meaning of conjunctive queries. A more formal definition is given in Section 3.13. In general, our definitions via first-order logic correspond to the Direct Semantics of OWL; for conjunctive queries under the RDF-Based Semantics, we provide some pointers in Section 6.

Summary Conjunctive query answering is a generalisation of instance retrieval. The main method to compute query answers in $\mathcal{QL}$*tiny* is query rewriting, explained below.

3.9 A Normal Form for $\mathcal{QL}$*tiny*

The rewriting of queries in $\mathcal{QL}$*tiny* is not too complicated, but it can be a bit cumbersome due to the syntactic form of $\mathcal{QL}$*tiny* axioms. To simplify our discussion, we will look at a slightly different ontology language that only allows axioms of the following forms:

QL normal form:

$$\begin{array}{ccc} A \sqsubseteq B & A \sqsubseteq \bot & A \sqsubseteq \exists P.B \\ A \sqcap A' \sqsubseteq B & \top \sqsubseteq B & \exists P.\top \sqsubseteq B \\ A(c) & P(c, d) & \end{array}$$

where A, A', and B are class names, and P is a property or an inverse property

We say that an axiom is in *QL normal form* if it is in one of these forms. In this section, we show how to transform $\mathcal{QL}$*tiny* ontologies into QL normal form. The idea of normalising axioms can be used in many situations to reduce the number of cases

If O contains an axiom of the form ...	then replace it by the set of axioms ...
$A^+(c)$	$\{F(c), F \sqsubseteq A^+\}$
$A^+ \sqsubseteq B^+$	$\{A^+ \sqsubseteq F, F \sqsubseteq B^+\}$
$A \sqsubseteq B_1 \sqcap B_2$	$\{A \sqsubseteq B_1, A \sqsubseteq B_2\}$
$A \sqsubseteq \exists P.B^+$	$\{A \sqsubseteq \exists P.F, F \sqsubseteq B^+\}$
$A \sqsubseteq \neg B$	$\{B \sqsubseteq F, A \sqcap F \sqsubseteq F', F' \sqsubseteq \bot\}$
$A \sqsubseteq \top$	$\emptyset$
$\bot \sqsubseteq B$	$\emptyset$

... where A^+ and B^+ must not be class names, A must be a class name, and F and F' always are fresh class names (not occurring in any axiom yet).

Fig. 3. Transformation of $\mathcal{QL}$*tiny* ontologies into QL normal form

that need to be considered in reasoning. Many implementations also compute suitable normal forms internally for that purpose.

Note that the QL normal form $A \sqcap A' \sqsubseteq B$ is not allowed in $\mathcal{QL}$*tiny*. Adding it does not make reasoning significantly more complex (see Remark 37), but it allows us to simplify our presentation of the reasoning algorithm.

Remark 37. Our addition of $\sqcap$ in subclasses of QL normal form does not lead to a significant increase in reasoning complexity. The resulting logic has a slightly larger combined worst-case complexity than OWL QL (namely P instead of NLogSpace), but is still tractable. For efficiently processing large amounts of assertional data (ABox), one is often more interested in the reasoning complexity that is obtained if we neglect the size of the TBox and query, that is, if we assume that they are bounded by some constant. This measure is called *data complexity* and is the same for OWL QL as for our extension with $\sqcap$ in subclasses. This might not have been clear when OWL QL has been designed: the slightest change of expressivity can lead to different computational properties, and there are a great number of feature combinations. A good overview is given in [1], where the authors study 40 different logics. In their terminology, our QL normal form is closely related to the logic $\textit{DL-Lite}^{\mathcal{H}}_{\textit{horn}}$ whereas the OWL QL is based on $\textit{DL-Lite}^{\mathcal{H}}_{\textit{core}}$.

QL normal form is expressive enough to capture all $\mathcal{QL}$*tiny* axioms. This is shown by providing a syntactic transformation algorithm:

Definition 38. Given a $\mathcal{QL}$*tiny* ontology O, an ontology $\mathsf{QLNF}(O)$ is obtained by exhaustively applying the replacement rules in Fig. 3 to axioms in O.

Example 39. To illustrate Definition 38, we use the axiom $\exists P^-.\top \sqsubseteq A \sqcap \exists Q.\exists R.B \sqcap \neg\exists S.\top$ as an abstract example. The rewriting steps are as follows, where we underline the axiom that is rewritten in each step:

$$\{\underline{\exists P^-.\top \sqsubseteq A \sqcap \exists Q.\exists R.B \sqcap \neg\exists S.\top}\}$$
$$\{\exists P^-.\top \sqsubseteq F_1, \underline{F_1 \sqsubseteq A \sqcap \exists Q.\exists R.B \sqcap \neg\exists S.\top}\}$$
$$\{\exists P^-.\top \sqsubseteq F_1, F_1 \sqsubseteq A, \underline{F_1 \sqsubseteq \exists Q.\exists R.B \sqcap \neg\exists S.\top}\}$$

$$\{\exists P^{-}.\top \sqsubseteq F_1, F_1 \sqsubseteq A, \underline{F_1 \sqsubseteq \exists Q.\exists R.B}, F_1 \sqsubseteq \neg\exists S.\top\}$$

$$\{\exists P^{-}.\top \sqsubseteq F_1, F_1 \sqsubseteq A, F_1 \sqsubseteq \exists Q.F_2, F_2 \sqsubseteq \exists R.B, \underline{F_1 \sqsubseteq \neg\exists S.\top}\}$$

$$\{\exists P^{-}.\top \sqsubseteq F_1, F_1 \sqsubseteq A, F_1 \sqsubseteq \exists Q.F_2, F_2 \sqsubseteq \exists R.B, \exists S.\top \sqsubseteq F_3, F_1 \sqcap F_3 \sqsubseteq F_4, F_4 \sqsubseteq \bot\}$$

The last line is the QL normal form that we wanted. Fresh class names F_1, F_2, F_3, and F_4 have been introduced to decompose axioms.

It is not hard to show that the transformation to QL normal form will always terminate in a linear number of steps. The steps for $A^{+}(c)$ and $A^{+} \sqsubseteq B^{+}$ can only be applied at most once to every axiom. Each of the remaining rules strictly decrease the nesting depth of operators in some class expression, or delete an axiom. The rewriting of $A \sqsubseteq B_1 \sqcap B_2$ is the only rule that duplicates an expression (A), but it is ensured that it is a class name only (so the total remaining work is not increased by the duplication).

Similarly, one can show that the result of the transformation is always in QL normal form. To do this, one has to observe that the transformation only leads to $\mathcal{QL}$*tiny* axioms (with the exception of case $A \sqsubseteq \neg B$, which also produces an axiom $A \sqcap F \sqsubseteq F'$ that is in QL normal form already and will not be transformed further). It remains to show that, for all $\mathcal{QL}$*tiny* axioms that are not in QL normal form, one of the rules of Definition 38 must apply. This can be shown by an easy case distinction.

Finally, one needs to show the semantic correctness of the transformation. We would like to be sure that conjunctive query answers over the original $\mathcal{QL}$*tiny* ontology are the same as query answers over the transformed ontology in QL normal form. This is true for all queries that use only vocabulary symbols that occurred in the original ontology, but none of the fresh auxiliary class names F that were introduced only in the introduction. Obviously, we cannot expect that these new classes F have the same meaning as in the original ontology (where they did not even occur).

Note that we need $\sqcap$ in subclasses only to express $\mathcal{QL}$*tiny* negation in QL normal form. Representing negation like this will be more convenient for our reasoning approach.

Summary $\mathcal{QL}$*tiny* ontologies can be transformed in linear time into ontologies in QL normal form with the same semantics. QL normal form allows arbitrary $\sqcap$ in subclasses.

3.10 Rewriting-Based Reasoning for $\mathcal{QL}$*tiny*

We are now ready to look at a concrete approach for answering conjunctive queries in $\mathcal{QL}$*tiny*. To simplify the presentation, we will usually not distinguish between properties and inverse properties. However, if P already is an inverse property R^{-}, then the expression P^{-} would be of the form $(R^{-})^{-}$, which is not allowed. Therefore, we assume $(R^{-})^{-}$ to be a shortcut for R in such a case.

The possible query rewriting steps are specified in Fig. 4 using the syntax of inference rules. However, the meaning of rules is slightly different than for our earlier calculi. A rule can be applied to a conjunctive query if the query contains the premises of the rule and the side conditions are satisfied. When we say that a query *contains* a

$$\mathbf{Q}_{\sqsubseteq}\ \frac{E(x)}{D(x)} : D \sqsubseteq E \in O \qquad\qquad \mathbf{Q}_{\mathsf{inv}}\ \frac{P(x,y)}{P^{-}(y,x)}$$

$$\mathbf{Q}^{-}_{\sqcap}\ \frac{D_1 \sqcap D_2(x)}{D_1(x) \quad D_2(x)} \qquad\qquad \mathbf{Q}^{-}_{\top}\ \frac{\top(x)}{}$$

$$\mathbf{Q}^{+}_{\exists}\ \frac{\exists P.\top(x) \quad \exists P^{-}.\top(y) \quad P(x,y) \quad P^{-}(y,x) \quad B(y)}{\exists P.B(x)} : \begin{array}{l}y \text{ a non-distinguished variable that occurs}\\ \text{only in the query atoms in the premise;}\\ \exists P.B \text{ occurs in } O\end{array}$$

plus any rule obtained from $\mathbf{Q}^{+}_{\exists}$ by leaving away some (but not all) of the premises

Fig. 4. Rewriting rules for conjunctive queries over ontologies in QL normal form

premise, we treat the query like a set of query atoms, i.e., we ignore the order and multiplicity of atoms. The application of a rule creates a new query, obtained by *replacing* the premise with the conclusion(s) of the rule.

Queries that are derived in this way are not supposed to replace the original query, but to be used in addition to find more answers. For example, rule $\mathbf{Q}_{\sqsubseteq}$ states that, to find all elements in E in the presence of an axiom $D \sqsubseteq E$, one also needs to find all elements in D. The subclass inclusion $D \sqsubseteq E$ can be any axiom in QL normal form. The remaining rules can be read in a similar way, but are based only on the meaning of class and property operators that does not depend on the presence of axioms in the ontology. Note that the queries that are created by such derivations may contain query atoms $D(x)$ where D is not just a class name. We allow this during the computation, and return to proper conjunctive queries later on.

$\mathbf{Q}^{+}_{\exists}$ is the only rule that builds more complex class expressions rather than decomposing them. This is necessary to be able to apply rule $\mathbf{Q}_{\sqsubseteq}$ for axioms of the form $A \sqsubseteq \exists P.B$. The side condition of $\mathbf{Q}^{+}_{\exists}$ ensures that all relevant information about y is captured by $\exists P.B(x)$. Moreover, we allow $\mathbf{Q}^{+}_{\exists}$ to be applied even if some of its premises are not given. In any case, however, the remaining premises must contain all occurrences of y. If only premises of the form $B(y)$ and $\exists P^{-}.\top(y)$ are given, then x does not occur in the premise. In this case, we use an arbitrary unused variable x in the conclusion. In contrast, B is determined by the side condition, even if it does not occur in the premise.

Example 40. As an example, consider the ontology from Example 35 and query (52). The rewriting rules lead to the following queries:

$\exists y.\mathit{FelisCatus}(x) \wedge \mathit{preysOn}(x,y) \wedge \mathit{Animal}(y)$	initial query	(53)
$\exists y.\mathit{FelisCatus}(x) \wedge \mathit{preysOn}^{-}(y,x) \wedge \mathit{Animal}(y)$	$\mathbf{Q}_{\mathsf{inv}}$: (53)	(54)
$\exists y.\mathit{FelisCatus}(x) \wedge \mathit{preysOn}(x,y) \wedge \mathit{SerinusCanaria}(y)$	$\mathbf{Q}_{\sqsubseteq}$: (53), (50)	(55)
$\exists y.\mathit{FelisCatus}(x) \wedge \mathit{preysOn}^{-}(y,x) \wedge \mathit{SerinusCanaria}(y)$	$\mathbf{Q}_{\mathsf{inv}}$: (55)	(56)
$\mathit{FelisCatus}(x) \wedge \exists \mathit{preysOn}.\mathit{Animal}(x)$	$\mathbf{Q}^{+}_{\exists}$: (53)	(57)
$\mathit{FelisCatus}(x)$	$\mathbf{Q}_{\sqsubseteq}$: (57), (51)	(58)

Note that we have slightly simplified queries during rewriting. We have dropped the existential quantifier after eliminating the non-distinguished variable y. Moreover, the step from (57) to (58) leads to an atom $\mathit{FelisCatus}(x)$ that is already present in the query; we have directly removed this duplicate (as mentioned before, we view conjunctions as sets during rewriting).

No further queries can be obtained through rewriting. For example, rule $\mathbf{Q}^+_\exists$ is not applicable to (55), since $\exists \mathit{preysOn}.\mathit{SerinusCanaria}$ does not occur in O. Moreover, $\mathbf{Q}^+_\exists$ cannot be applied to a proper subset of the premise $\mathit{preysOn}(x, y)$, $\mathit{Animal}(y)$, since y would occur outside this set. Finally, applying $\mathbf{Q}^+_\exists$ to (54) would be possible, but only to obtain query (57) again.

To find all answers to the original query, each of the rewritten queries are answered over the ABox, and the results are combined. When doing this, we do not need to consider any queries that contain classes of the form $\exists P.B$ with $B \neq \top$, since they must have been introduced using rule $\mathbf{Q}^+_\exists$, and the premises of this rule are equivalent to the conclusion. Likewise, we do not need queries with classes of the form $A_1 \sqcap A_2$, which are decomposed using rule $\mathbf{Q}^-_\sqcap$.

The remaining queries may still contain expressions of the form $\exists P.\top(x)$. We rewrite these expressions to $P(x, y)$ where y is a fresh variable not used in the query yet. This yields the final set of queries that we need to answer over the ABox. At this stage, it does not matter any more whether a variable is distinguished or non-distinguished, since the ABox contains only information about individual names anyway.

Example 41. Of the queries obtained in Example 40, queries (53), (54), (55), (56), and (58) are relevant to answer the original query. Further analysis could be used to recognise that all of these queries are subsumed by (58) (i.e., have at most the answers of (58)). Therefore, one can find all results of the original query by evaluating (58) only. For the ABox of Example 35, this yields the two results $x = \mathit{silvester}$ and $x = \mathit{tom}$, as expected.

To practically answer the queries over the ABox, we could, for example, compute the possible answers for every query atom individually and then combine (join) the results. This part is standard query answering over a database (conjunctive queries can easily be expressed in query languages like SQL or SPARQL), and many optimised procedures are available. However, in general, the computation of query answers is exponential in the size of the query, which is the case even for databases without any additional ontology. In addition, the number of queries that are obtained by rewriting can also be exponential. As mentioned in Remark 37, reasoning for QL normal form is tractable and even highly scalable with respect to the size of the ABox. This does not mean that our algorithm actually achieves these worst-case optimal bounds: as the following remark discusses, it shows the latter but not the former.

Remark 42. For some inputs in QL normal form, our query rewriting algorithm may produce exponentially many rewritten queries, even for instance retrieval queries (those that have only one query atom). To give an example, we consider axioms of the form $C_{w0} \sqcap C_{w1} \sqsubseteq C_w$ for all words w in $\{0, 1\}^*$ of length at most ℓ. For example, if $\ell = 2$, we obtain:

$$C_0 \sqcap C_1 \sqsubseteq C \quad C_{00} \sqcap C_{01} \sqsubseteq C_0 \quad C_{10} \sqcap C_{11} \sqsubseteq C_1 \quad C_{000} \sqcap C_{001} \sqsubseteq C_{00} \quad \ldots \quad C_{110} \sqcap C_{111} \sqsubseteq C_{11}$$

There are $2^{\ell+1} - 1$ such axioms (7 in our example). Now the query $C(x)$ has (among others) a rewriting of the form $\bigwedge_{w\in\{0,1\}^\ell} C_w(x)$. In our example:

$$C_{00}(x) \land C_{01}(x) \land C_{10}(x) \land C_{11}(x)$$

This rewriting has 2^ℓ atoms (4 in our example). Now each of these atoms $C_w(x)$ could again be rewritten into two atoms $C_{w0}(x) \land C_{w1}(x)$. There are 2^{2^ℓ} many possible rewritings of this form (16 in our example). For example, we can obtain:

$$\begin{aligned} C_{00}(x) \land C_{01}(x) \land C_{10}(x) \land C_{11}(x) \\ C_{000}(x) \land C_{001}(x) \land C_{01}(x) \land C_{10}(x) \land C_{11}(x) \\ C_{00}(x) \land C_{010}(x) \land C_{011}(x) \land C_{10}(x) \land C_{11}(x) \quad \ldots \end{aligned}$$

Thus, we have an ontology of $s = 2^{\ell+1} - 1$ axioms for which we get at least $2^{(s+1)/2}$ query rewritings – that is, exponentially many. This shows that our algorithm is not polynomial.

This is normal (and unavoidable) when rewriting conjunctive queries, where each query atom might either be rewritten or not. In our case, however, we start with a simple query $C(x)$ and use $\sqcap$ in subclasses to introduce more and more query atoms. Going back to the transformation to QL normal form in Fig. 3, it is not hard to see that a $\mathcal{QL}$*tiny* input cannot lead to a situation as in the problematic example above: $\sqcap$ in subclasses is only used with fresh class names in superclasses, which cannot be "layered" as above. Indeed, for $\mathcal{QL}$*tiny* inputs, our algorithm is polynomial.

For general QL normal form ontologies, our algorithm thus does not show tractability of reasoning. However, it does show the low *data* complexity that we claimed above, since the number of rewritings does not matter in this case (it would be considered constant).

Finally, we can observe why this approach cannot work for $\mathcal{EL}$*tiny* and $\mathcal{RL}$*tiny*:

Example 43. Consider an ontology that contains the following TBox axiom

$$\exists \mathit{hasMother}.\mathit{Human} \sqsubseteq \mathit{Human}$$

which states that everybody who has a human mother must also be human. Then an element x is an answer to the query $\mathit{Human}(x)$ whenever it matches any of the following queries:

$$\begin{aligned} &\mathit{Human}(x) \\ \exists y_1.&\mathit{hasMother}(x, y_1) \land \mathit{Human}(y_1) \\ \exists y_1, y_2.&\mathit{hasMother}(x, y_1) \land \mathit{hasMother}(y_1, y_2) \land \mathit{Human}(y_2) \\ \exists y_1, y_2, y_3.&\mathit{hasMother}(x, y_1) \land \mathit{hasMother}(y_1, y_2) \land \mathit{hasMother}(y_2, y_3) \land \mathit{Human}(y_3) \\ &\ldots \end{aligned}$$

In other words, an ABox from which it follows that x is human may involve an arbitrarily long chain of *hasMother* relations. Without knowing the ABox in advance, there is no way of restricting the number of rewritten queries. This explains why existential quantifiers in $\mathcal{QL}$*tiny* subclasses can only use $\top$ as a filler.

Summary Query rewriting for QL normal form ontologies is done by applying rewriting rules until all possible queries have been computed. The (post-processed) queries are then evaluated over the ABox.

3.11 Completing the Query Rewriting Method

The query rewriting method introduced above can already be used to obtain correct query answers, but it is not complete yet. To fix this, two additional considerations are necessary. First, we have obviously neglected the possibility that the ontology is inconsistent. In this case, every possible solution mapping should be an answer to every query. Second, our side conditions for rule $\mathbf{Q}_{\exists}^{+}$ prevent its application in some cases where a query condition is essentially duplicated. To solve this, we need to allow variables to be *unified* during rewriting.

Taking inconsistency into account turns out to be very simple. We merely need to use our approach to answer the query $\exists y.\bot(y)$. Again, this is not really a valid conjunctive query, since $\bot$ is not a class name, but the rewriting may lead to proper queries. If any of these queries has an answer, then the ontology is inconsistent. This needs to be checked before answering any other queries. Note that y is non-distinguished in the query $\exists y.\bot(y)$, so the answer will only be "match" or "no match" without giving any variable bindings. Such queries are called *Boolean queries*.

To understand the second problem, consider the following example:

Example 44. Consider the following ontology:

$$\mathit{Predator}(\mathit{silvester}) \qquad \mathit{Predator} \sqsubseteq \exists \mathit{preysOn}.\mathit{Animal} \qquad \mathit{Animal} \sqsubseteq \exists \mathit{hasMother}.\mathit{Animal}$$

and the query

$$\exists y_1, y_2, z.\mathit{preysOn}(x, y_1) \wedge \mathit{hasMother}(y_1, z) \wedge \mathit{preysOn}(x, y_2) \wedge \mathit{hasMother}(y_2, z). \tag{59}$$

This query is basically just a complicated way of stating the query

$$\exists y_1, z.\mathit{preysOn}(x, y_1) \wedge \mathit{hasMother}(y_1, z). \tag{60}$$

Indeed, there is no reason why y_1 and y_2 need to represent different elements. So whenever the query atoms in query (60) are satisfied, the additional atoms in (59) that involve y_2 can also be satisfied. Therefore, both queries have the same answer $x = \mathit{silvester}$. Indeed, query (60) can be rewritten as follows:

$$\begin{array}{lr}
\exists y_1, z.\mathit{preysOn}(x, y_1) \wedge \mathit{hasMother}(y_1, z) & \\
\quad \exists y_1.\mathit{preysOn}(x, y_1) \wedge \exists \mathit{hasMother}.\mathit{Animal}(y_1) & \mathbf{Q}_{\exists}^{+} \\
\quad \exists y_1.\mathit{preysOn}(x, y_1) \wedge \mathit{Animal}(y_1) & \mathbf{Q}_{\sqsubseteq} \\
\qquad \exists \mathit{preysOn}.\mathit{Animal}(x) & \mathbf{Q}_{\exists}^{+} \\
\qquad \mathit{Predator}(x) & \mathbf{Q}_{\sqsubseteq}
\end{array}$$

The final query then yields the desired result. However, query rewriting does not work as expected for (59). The reason is that we cannot apply rule $\mathbf{Q}_{\exists}^{+}$, since the variable z occurs in multiple atoms that cannot be eliminated in one application of this rule. Therefore, the only possible rewritings are based on rule $\mathbf{Q}_{\mathbf{inv}}$, and none of the rewritten queries has any results over the ABox.

To fix this problem, we introduce an additional query transformation operation that can be applied during rewriting: *factorisation*. It is applicable to a query Q if the following conditions hold:

– Q contains conditions $\mathsf{P}(x, y)$ and $\mathsf{P}(x', y)$,
– $x \neq x'$,
– x is distinguished or x' is non-distinguished.

In this case, a new query Q' is obtained by replacing every occurrence of x' in Q by x.

Example 45. Query (59) in Example 44 can be factorised by mapping y_2 to y_1. The result is the same as query (60), provided that we eliminate duplicate atoms.

If distinguished variables are eliminated in factorisation, then this needs to be taken into account for computing the final results. It is obvious how to do this: if a distinguished variable x was replaced by a (necessarily also distinguished) variable y, then the variable assignments found for y will also count as assignments for x in the final query evaluation.

Summary To obtain a complete query answering procedure for QL normal form ontologies, query rewriting must check inconsistencies and support factorisation.

3.12 Correctness of Query Rewriting

It remains to show that query rewriting leads to a correct reasoning method for $\mathcal{QL}$*tiny*. The properties we need to verify are the same as before: well-definedness, termination, soundness, and completeness.

Well-definedness For this we merely need to note that the rewriting of queries is again a monotone saturation process. Although we replace (delete and insert) atoms in an individual rewriting step, the resulting query is always *added* to the overall result set. Every rewriting step increases the set of rewritten queries, and additional queries never prevent a rewriting step from being possible. Therefore, the process must eventually lead to *saturation*.

Termination It is essential for termination that we eagerly eliminate duplicate query atoms. Moreover, we need to abstract from the names of non-distinguished variables, i.e., two queries that only differ in the name of some non-distinguished variable must be considered to be the same. The reason for termination is then as follows:

– in every rewriting step, the total number of variables in the query remains the same or becomes smaller;
– all class expressions that occur in a rewritten query occur either in the initial query or in the ontology;
– every property expression that occurs in a rewritten query is one that occurs in the initial query, an inverse thereof, or a property expression that occurs in the ontology.

These properties are preserved by each rule application, and can therefore be shown by an induction over the length of the derivation. Thus, the total number of classes and properties that can be used in query atoms are bounded linearly by the size of the ontology and the initial query, and the maximal number of variables per query is bounded by a constant. Up to renaming of variables, there is only a finite (but exponential) number of different queries that can be constructed from this vocabulary. Therefore, the rewriting must terminate.

Soundness Soundness can be established as in the case of $\mathcal{RL}$*tiny* and $\mathcal{EL}$*tiny* by noting that every rule is sound. The exact property in this case is: if a variable assignment is an answer to the query that was produced by applying a rewriting rule, then it is also an answer to the query that has been rewritten in this step, given that we take the ontology into account. This is easy to verify for each rule. The overall claim can be shown by an inductive argument over the length of the derivation.

Completeness, once more, is more difficult to show. We will lay out the main ideas of this proof in the next section.

Summary Well-definedness, termination, and soundness of query rewriting are not hard to show using similar ideas as for the saturation approaches used in $\mathcal{RL}$*tiny* and $\mathcal{EL}$*tiny*.

3.13 Completeness of Query Rewriting

To show completeness of query rewriting, we need a couple of additional techniques. Like in the case of $\mathcal{RL}$*tiny* and $\mathcal{EL}$*tiny*, we make use of a canonical model, but now this model turns out to be infinite. Moreover, we need to be somewhat more precise about the semantics of query answering. In this section, we outline the basic argumentation used for this proof, since it is interesting in its own right (and different from what we have seen so far). Readers who want to focus on the practical aspects of the OWL profiles, however, can safely skip this section.

Semantics of Query Answering Let us first introduce some additional notation to formalise the semantics of conjunctive query answering, which we have only discussed in an intuitive fashion so far. A *solution mapping* σ for a conjunctive query Q is a substitution that maps each distinguished variable x of Q to an individual name $x\sigma$. We write $Q\sigma$ to denote the formula obtained by replacing every distinguished variable x in Q by $x\sigma$. Therefore, $Q\sigma$ is a first order formula of the form $\exists \boldsymbol{y}.A_1 \wedge \ldots \wedge A_\ell$, where $\boldsymbol{y}$ is a list of variables, and each atom A_i is a formula of the form $\mathsf{C}(t)$ or $\mathsf{P}(t, s)$ and t, s are either variables from $\boldsymbol{y}$ or individual names.

The first-order semantics of existentially quantified variables is defined as usual by means of *variable assignments*. Given an interpretation $\mathcal{I}$, a variable assignment $\mathcal{Z}$ for $\mathcal{I}$ is a function from variables to domain elements of $\Delta^{\mathcal{I}}$. If t is an individual name or variable, then $t^{\mathcal{I},\mathcal{Z}} = t^{\mathcal{I}}$ if t is an individual name, and $t^{\mathcal{I},\mathcal{Z}} = \mathcal{Z}(t)$ if t is a variable. According to the standard semantics of first-order logic, an interpretation $\mathcal{I}$ *satisfies* $Q\sigma$ if there is a variable assignment $\mathcal{Z}$ such that:

– for each query atom $\mathsf{C}(t)$ in $Q\sigma$, we have $t^{\mathcal{I},\mathcal{Z}} \in \mathsf{C}^{\mathcal{I}}$;

- for each query atom $\mathsf{P}(t, s)$ in $Q\sigma$, we have $\langle t^{\mathcal{I},\mathcal{Z}}, s^{\mathcal{I},\mathcal{Z}}\rangle \in \mathsf{P}^{\mathcal{I}}$.

The formula $Q\sigma$ is entailed by an ontology $\mathcal{O}$, written $\mathcal{O} \models Q\sigma$, if all models of $\mathcal{O}$ satisfy $Q\sigma$. In this case, the solution mapping σ is an *answer* for Q with respect to $\mathcal{O}$. In essence, these remarks merely recall some standard ideas of first-order logic to formalise what we said earlier.

Defining a Canonical Model We now define a canonical model for ontologies in QL normal form. For now, we consider only ontologies that do not contain the empty class $\bot$; it will come in only later. The construction of the canonical model for such ontologies is very similar to the saturation process used in $\mathcal{RL}$*tiny* reasoning. However, instead of deriving ABox facts, we now apply class inclusion axioms directly to interpretations:

Definition 46. Consider an interpretation $\mathcal{I}$ with domain $\Delta^{\mathcal{I}}$, and an axiom $\mathsf{C} \sqsubseteq \mathsf{D}$ in QL normal form with $\mathsf{D} \neq \bot$. We say that $\mathsf{C} \sqsubseteq \mathsf{D}$ is *applicable* to an element $e \in \Delta^{\mathcal{I}}$ if $e \in \mathsf{C}^{\mathcal{I}}$. In this case, we can *apply* $\mathsf{C} \sqsubseteq \mathsf{D}$ to $\mathcal{I}$ to obtain a new interpretation $\mathcal{J}$ that is defined as follows:

(a) If D is a class name, then $\mathsf{D}^{\mathcal{J}} = \mathsf{D}^{\mathcal{I}} \cup \{e\}$.
(b) If $\mathsf{D} = \exists \mathsf{P}.\mathsf{B}$, then $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \{f\}$ for a fresh element $f \notin \Delta^{\mathcal{I}}$, $\mathsf{B}^{\mathcal{J}} = \mathsf{B}^{\mathcal{I}} \cup \{f\}$, and $\mathsf{P}^{\mathcal{J}} = \mathsf{P}^{\mathcal{I}} \cup \{\langle e, f\rangle\}$ (this is understood even if P is inverse).

In all other aspects, $\mathcal{J}$ agrees with $\mathcal{I}$.

Intuitively speaking, applying an axiom to an interpretation means that we modify the interpretation in such a way that an element e that is in the subclass of the axiom must also be in the superclass of the axiom. We can now build an interpretation by applying rules in a (possibly infinite) construction.

Definition 47. Consider an ontology $\mathcal{O}$ with axioms in QL normal form. An initial interpretation $\mathcal{I}_0$ is defined as follows:

- The domain $\Delta_0^{\mathcal{I}}$ of $\mathcal{I}_0$ is the set of all individual symbols in the signature (without loss of generality, we can assume that there is at least one, even if it does not occur in $\mathcal{O}$).
- For every individual symbol c, define $\mathsf{c}^{\mathcal{I}_0} := \mathsf{c}$.
- For every class name A, define $\mathsf{c} \in \mathsf{A}^{\mathcal{I}_0}$ if and only if $\mathsf{A}(\mathsf{c}) \in \mathcal{O}$.
- For every property name P, define $\langle \mathsf{c}, \mathsf{d}\rangle \in \mathsf{P}^{\mathcal{I}_0}$ if and only if $\mathsf{P}(\mathsf{c}, \mathsf{d}) \in \mathcal{O}$ or $\mathsf{P}^{-}(\mathsf{d}, \mathsf{c}) \in \mathcal{O}$.

Given an interpretation $\mathcal{I}_n$, the interpretation $\mathcal{I}_{n+1}$ is constructed by applying an axiom of $\mathcal{O}$ to an element of $\mathcal{I}_n$. If this is not possible, then $\mathcal{I}_{n+1} = \mathcal{I}_n$.

This process defines an interpretation $\mathcal{I}_n$ for every natural number n. However, the construction is not deterministic, since there might be many elements to which some axiom can be applied in each step. To fix this, we need to fix a strategy for applying axioms to elements. There are many ways to do this; one is sketched in Remark 48.

Remark 48. To fix a deterministic strategy of constructing the interpretations $\mathcal{I}_n$ in Definition 47, we can assume that all axioms are applied in some (fixed but arbitrary order). To this end, assume that the axioms of the ontology are totally ordered, and will always be applied in that order. Likewise, assume that all individual names have a total order, and that, whenever we introduce a fresh domain element, we define it to be bigger than all existing elements with respect to this order. This will ensure that new elements are only processed *after* all current elements have been considered. Now we can apply rules with the following strategy:

(1) Initialise e be the largest domain element
(2) Apply all applicable axioms to e (in the order chosen for axioms)
(3) Update e: if e is the smallest domain element, set e to be the largest domain element;
otherwise, set e to be the next largest domain element below the current e
(4) Go to (2)

This defines a fair strategy for applying axioms in Definition 47 (*fair* means that every axioms that is applicable to some element e will actually be applied at some point in time). To make the process fully deterministic, we should additionally fix what the new elements f in step (b) of Definition 46 are, although this does not have an effect on the semantics. For example, we could use natural numbers for f, always picking the smallest number that is not in the domain of $\mathcal{I}$ yet.

This allows us to assume that each $\mathcal{I}_n$ is uniquely determined. Now we can define $\mathcal{I}$ to be the union of all (infinitely many) interpretations $\mathcal{I}_n$:[9]

- The domain of $\mathcal{I}$ is $\Delta^{\mathcal{I}} = \bigcup_{n\geq 0} \Delta_n^{\mathcal{I}}$.
- For every individual symbol c, we set $c^{\mathcal{I}} := c^{\mathcal{I}_0}$.
- For every class or property name X, we set $X^{\mathcal{I}} = \bigcup_{n\geq 0} X_n^{\mathcal{I}}$.

This defines the *canonical model* $\mathcal{I}$ of the original ontology O. $\mathcal{I}$ is usually infinite – already a single axiom $A \sqsubseteq \exists P.A$ suffices to create infinitely many domain elements. This is not a concern for us, since we only need $\mathcal{I}$ as a theoretical guideline for showing completeness. The canonical quality of $\mathcal{I}$ is expressed in the following statement:

Theorem 49. Consider an ontology O in QL normal form that does not contain $\bot$, and let $\mathcal{I}$ be the canonical model of O. Then $\mathcal{I}$ is a model of O.

Moreover, the answers of a conjunctive query over O are the same as the answers of Q over $\mathcal{I}$. More precisely, if σ is a solution mapping for the distinguished variables of Q, then $O \models Q\sigma$ if and only if $\mathcal{I} \models Q\sigma$.

We do not go into the details of proving this statement. The key idea for the proof is that the relationships that hold in the canonical model must hold in every model of O. So whenever a query matches the canonical model, it also matches every other model.

Showing Completeness We can use the canonical model and its construction to show completeness of query rewriting for ontologies without $\bot$.

[9] On a first glance, the union might seem unnecessary, given that the models $\mathcal{I}_n$ are growing monotonically ("$\mathcal{I}_n \subseteq \mathcal{I}_{n+1}$"). However, since this is an infinite union, we can generally not write it as $\mathcal{I}_k$ for any natural number k. The union is a formal way of writing "$\mathcal{I}_\infty$".

Theorem 50. Consider an ontology O in QL normal form without $\bot$. Let Q be a conjunctive query and let σ be a solution mapping for Q. If $\mathsf{O} \models Q\sigma$ then there is a query rewriting P of Q such that $P\sigma$ is entailed by the ABox of O.

Proof. We show the claim for generalised conjunctive queries Q that may also contain query atoms of the form $\exists \mathsf{P}.\top(x)$ but no other complex class expressions (i.e., $A_1 \sqcap A_2(x)$ or $\exists \mathsf{P}.\mathsf{B}(x)$).

Consider a query answer σ as in the claim, and let $\mathcal{I}$ be the canonical model of O. By Theorem 49, $\mathcal{I} \models Q\sigma$. Since Q is finite, there is a number n such that $\mathcal{I}_n \models Q\sigma$. Let n be the smallest number with this property. We show the claim by induction over n.

If $n = 0$, then, according to the construction of $\mathcal{I}_0$, $Q\sigma$ is entailed by the ABox of O. This establishes the claim.

If $n > 0$, then $\mathcal{I}_{n-1} \not\models Q\sigma$ since n was chosen to be the smallest number with $\mathcal{I}_n \models Q\sigma$. Since $\mathcal{I}_n \models Q\sigma$, there is a variable assignment $\mathcal{Z}$ for $\mathcal{I}_n$ under which all query atoms in $Q\sigma$ are satisfied (we will refer to this particular assignment $\mathcal{Z}$ below). By definition, $\mathcal{I}_n$ has been obtained from $\mathcal{I}_{n-1}$ by applying some axiom $\mathsf{C} \sqsubseteq \mathsf{D}$ to some element $e \in \Delta^{\mathcal{I}}_{n-1}$. We need to consider the two possible cases of Definition 46:

Case (a) The only change from $\mathcal{I}_{n-1}$ to $\mathcal{I}_n$ is that the extension of D has been extended to include e. Thus, since $\mathcal{I}_{n-1} \not\models Q\sigma$, Q must contain at least one atom of the form $\mathsf{D}(x)$ where $(x\sigma)^{\mathcal{I},\mathcal{Z}} = e$. Using rule $\mathbf{Q}_{\sqsubseteq}$, we can obtain a query Q' where all such atoms $\mathsf{D}(x)$ are replaced by $\mathsf{C}(x)$. The query Q'' is obtained from Q' by exhaustively applying $\mathbf{Q}^{-}_{\sqcap}$ and $\mathbf{Q}^{-}_{\top}$ (to cover the case where C contains $\sqcap$ or $\top$; recall that we do not allow these in the queries for which we prove the claim). By assumption, $e \in \mathsf{C}^{\mathcal{I}_{n-1}}$, so $\mathcal{I}_{n-1} \models Q''\sigma$ can be shown by the same variable assignment $\mathcal{Z}$ that showed $\mathcal{I}_n \models Q\sigma$. By the induction hypothesis, there is a rewriting P of Q'' for which σ is an answer over the ABox. Since P is also a rewriting of Q, this shows the claim.

Case (b) Then $\mathsf{D} = \exists \mathsf{P}.\mathsf{B}$ and $\mathcal{I}_n$ extends $\mathcal{I}_{n-1}$ with an additional domain element f and according relationships. The query atoms that we allow in Q can capture exactly the following semantic conditions that hold in $\mathcal{I}_n$ but (possibly) not in $\mathcal{I}_{n-1}$:

$$e \in \exists \mathsf{P}.\top^{\mathcal{I}_n} \qquad f \in \exists \mathsf{P}^{-}.\top^{\mathcal{I}_n} \qquad \langle e, f\rangle \in \mathsf{P}^{\mathcal{I}_n} \qquad \langle f, e\rangle \in (\mathsf{P}^{-})^{\mathcal{I}_n} \qquad f \in \mathsf{B}^{\mathcal{I}_n} \tag{61}$$

Since $\mathcal{I}_{n-1} \not\models Q\sigma$, Q must thus contain at least one query atom of the following forms:

$$\exists \mathsf{P}.\top(x) \qquad \exists \mathsf{P}^{-}.\top(y) \qquad \mathsf{P}(x, y) \qquad \mathsf{P}^{-}(y, x) \qquad \mathsf{B}(y) \tag{62}$$

where $(x\sigma)^{\mathcal{I},\mathcal{Z}} = e$ and $(y\sigma)^{\mathcal{I},\mathcal{Z}} = f$. Moreover, Q cannot contain any other atoms of the form $\mathsf{E}(y)$, since they would not be satisfied in $\mathcal{I}_n$. Likewise, all atoms of the form $\mathsf{P}(x, y)$ and $\mathsf{P}^{-}(y, x)$ must be such that $(x\sigma)^{\mathcal{I},\mathcal{Z}} = e$. Yet, there can be multiple such variables x in different atoms, e.g., $\mathsf{P}^{-}(y, x_1)$ and $\mathsf{P}(x_2, y)$, as long as $(x_1\sigma)^{\mathcal{I},\mathcal{Z}} = (x_2\sigma)^{\mathcal{I},\mathcal{Z}} = e$. Applying rule $\mathbf{Q}_{\mathsf{inv}}$, we can obtain a query Q' that contains an atom $\mathsf{P}(x, y)$ for every atom $\mathsf{P}^{-}(x, y)$. If multiple variables x_i occur in such atoms, we can identify them using factorisation.

Thus, we obtain a query Q'' where there is at most one statement of the form $\mathsf{P}(x, y)$ (and at most one statement of the form $\mathsf{P}^{-}(y, x)$) for every variable y with $(y\sigma)^{\mathcal{I},\mathcal{Z}} = f$.

Therefore, rule $\mathbf{Q}^{+}_{\exists}$ can be applied to replace all sets of atoms of the form (62) by the atom $\exists P.B(x)$. As explained before, we may introduce a new non-distinguished variable x for that purpose if it is not determined by the premises. In this case, we extend the assignment $\mathcal{Z}$ by setting $\mathcal{Z}(x) = e$. Finally, we apply rule $\mathbf{Q}_{\sqsubseteq}$ to replace all occurrences of $\exists P.B(x)$ by $C(x)$, and exhaustively rewrite the resulting query using $\mathbf{Q}^{-}_{\sqcap}$ and $\mathbf{Q}^{-}_{\top}$.

If Q''' is the result of this transformation, then $\mathcal{I}_{n-1} \models Q'''\sigma$ holds since $e \in C^{\mathcal{I}_{n-1}}$ as in Case (a). The variable assignment $\mathcal{Z}$ can be used to show this, since we extended $\mathcal{Z}$ to cover new variables introduced in the transformation. Assignments $\mathcal{Z}(y) = f$ can simply be forgotten (all such variables y have been eliminated). The overall claim follows by induction as in Case (a). □

Adding $\bot$ and Wrapping up Theorem 50 already shows completeness for all ontologies in OWL normal form that do not contain $\bot$. To cover the cases with $\bot$, we only need a very simple transformation: given an ontology O, the ontology O' is obtained by replacing all occurrences of $\bot$ with a new class name $B_{\bot}$ that was not used anywhere yet. Then O is inconsistent if and only if $O' \cup \{B_{\bot} \sqsubseteq \bot\}$ is. Clearly, this is only the case if $B_{\bot}$ is inferred to be non-empty, which we can check by evaluating the query $\exists y.B_{\bot}(y)$ over O'. Theorem 50 shows that this check is complete. It is clear that the results of this check agree with the results of the check $\exists y.\bot(y)$ on O.

Provided that O is consistent, all conjunctive queries that do not use $B_{\bot}$ have the same answers over O and O'. Thus, the answers to such queries can again be obtained completely according to Theorem 50.

Summary Completeness of query answering for QL normal forms (and thus for $\mathcal{QL}$*tiny*) can be shown with the help of an infinite canonical model.

4 The Limits of Lightweight Ontology Languages

So far, we have seen various sublanguages of OWL (or rather: $\mathcal{ALCI}$) for which reasoning was possible in polynomial time. Could we add even more features without loosing this useful property? Is it even justified to have three different profiles, or could we just combine the profiles to obtain a single lightweight ontology language that subsumes them all? In the following sections, we explain how to investigate such questions. We will see that some extensions of the profiles are indeed feasible, but that many others make reasoning exponentially harder.

4.1 What Really Counts: Relaxing Unnecessary Restrictions

Throughout Section 3, we have seen a number of techniques for solving typical reasoning problems for our tiny versions of the three OWL profiles. Our goal was to explain and illustrate the most essential tools we have for reasoning with lightweight ontology languages today. The natural question now is: how strongly do these techniques depend on the tiny languages that we have picked to illustrate them? This section shows how far we can go with our approaches (at the very least). This will help us to understand

CL	**CName**	$\top$	$\bot$	**CL** $\sqcap$ **CL**	**CL** $\sqcup$ **CL**	$\neg$**CR**	$\exists$**P**.**CL**	$\exists$**P**.$\top$	$\forall$**P**.**CL**	$\forall$**P**.$\bot$
OWL RL	×	(×)	×	×	×		×	×		
OWL EL	×	×	×	×	(×)		×	×		
OWL QL	×	×	×	(×)	(×)			×		

CR	**CName**	$\top$	$\bot$	**CR** $\sqcap$ **CR**	**CR** $\sqcup$ **CR**	$\neg$**CL**	$\exists$**P**.**CR**	$\exists$**P**.$\top$	$\forall$**P**.**CR**	$\forall$**P**.$\bot$
OWL RL	×	(×)	×	×		×			×	×
OWL EL	×	×	×	×		(×)	×	×		(×)
OWL QL	×	×	×	×		×	×	×		(×)

Fig. 5. $\mathcal{ALCI}$ class expressions allowed on the left (top) and right (bottom) of subclass inclusions in the OWL profiles; parentheses mean that a feature is not in a profile but could easily be added

which differences between the profiles are really essential for implementing efficient systems.

The definitions of our three tiny ontology languages have been restricted for three different and partly independent reasons:

(1) Reasoning complexity: allow lightweight reasoning methods to work
(2) Standard compliance: make sure that each language is contained in its corresponding OWL profile
(3) Didactic convenience: disregard inessential features to simplify the presentation

From a user perspective, only (1) is really a good reason. Standard compliance is of course important, but it is not against the standard if a tool supports a few additional features.[10] And indeed, every OWL profile has some omissions that are not motivated by (1). Item (3) might actually be important in some applications, too. For example, OWL EL has been deliberately restricted to simplify its definition (as opposed to RL and QL, it does not require different definitions for subclass and superclass expressions). However, we should still have a clear understanding if an omitted feature is merely deemed too hard to understand for users, or if it would actually derail our whole reasoning approach and possibly boost the overall computational complexity.

Even when restricting to the features in $\mathcal{ALCI}$, our tiny profiles can be significantly extended without loosing their good properties for reasoning. The axioms of the extended languages can in each case be described as follows:

$$\mathbf{Axiom} ::= \mathbf{CL} \sqsubseteq \mathbf{CR} \mid \top \sqsubseteq \forall \mathbf{P}.\mathbf{CR} \mid \mathbf{CR}(\mathbf{IName}) \mid \mathbf{P}(\mathbf{IName}, \mathbf{IName}),$$

where the definitions of **CL** and **CR** for each profile are shown in Fig. 5. As before **P** includes inverse properties for OWL RL and OWL QL, but not for OWL EL. Note that we generally allow range axioms $\top \sqsubseteq \forall \mathbf{P}.\mathbf{CR}$ which had only been allowed in $\mathcal{RL}$*tiny*

[10] Actually, a reasoner can even conform to the OWL standard if it does not support all features of OWL or one of its profiles; the important quality is that the tool knows about its limitations and does not accidentally give wrong answers.

so far. Indeed, all OWL profiles support ranges using a special syntax. In $\mathcal{QL}$*tiny* this is syntactic sugar that could be transformed away during normalisation (as Example 21 illustrated), but for $\mathcal{EL}$*tiny* the saturation algorithm needs to be extended to support ranges properly. Moreover, some advanced features of OWL EL (property chains; see Section 5) are restricted in the presence of range axioms.

Let us now take a closer look at the features in Fig. 5. Crosses indicate that a feature is directly supported in the respective OWL profile, while crosses in parentheses mean that a feature could easily be added without major changes in our reasoning algorithms. We specifically list $\exists\mathbf{P}.\top$ due to its special role in QL, and (mainly for symmetry) we also mention $\forall\mathbf{P}.\bot$.

At a first glance, it is remarkable how similar the profiles are – or rather: could be – regarding most features. For example, the expressions **CL** on the left of subclass inclusions are essentially the same in all cases. The only relevant difference is the omission of $\exists\mathbf{P}.\mathbf{CL}$ in OWL QL. We have seen in Example 43 why this restriction is important for query rewriting. Unions $\sqcup$ on the left can always be allowed, either directly during reasoning as we did for $\mathcal{RL}$*tiny*, or by transforming them away in a preprocessing step similar to the normalisation of $\mathcal{QL}$*tiny* in Section 3.9. Indeed, $A \sqcup B \sqsubseteq C$ can always be written as $A \sqsubseteq C$, $B \sqsubseteq C$. This also means that this feature is always syntactic sugar that does not allow any new relationships to be modelled.

In contrast, intersection $\sqcap$ on the left cannot be expressed indirectly in an easy way. Including it in OWL QL would really improve its expressivity. We have already specified a reasoning method for OWL QL that achieves low data complexity in the presence of intersections; some more would be needed to obtain a procedure that is also tractable with respect to the size of the ontology (see Remark 42). Allowing $\top$ in OWL RL is also a slight extension, since it cannot be expressed indirectly. However, OWL RL already allows $\top$ in expressions $\exists\mathbf{P}.\top$ (which we did not include in $\mathcal{RL}$*tiny*). Allowing $\top$ everywhere would mainly simplify the language description, and not really add practically relevant modelling power.

Considering the right-hand classes **CR**, we see that the Boolean features are again the same in all cases. Adding $\neg\mathbf{CL}$ in OWL EL is easy to do, since it is again syntactic sugar. Our transformation of $\mathcal{QL}$*tiny* to QL normal form showed how one can eliminate such negations using $\sqcap$ and $\bot$. The addition of $\top$ in OWL RL is again only a formal simplification, as $\top$ is never relevant as a superclass. The main difference between the profiles thus is in their support of quantifiers in superclasses. OWL RL supports universal quantifiers while OWL EL and QL support existentials. The special case of $\forall\mathbf{P}.\bot$ can be used to state that no **P**-relations exist. Axioms of the form $C \sqsubseteq \forall\mathbf{P}.\bot$ can be expressed as $C \sqsubseteq B$, $B \sqcap \exists\mathbf{P}.\top \sqsubseteq \bot$ using a new class name B. This feature is therefore syntactic sugar in all profiles (provided that we allow for $\sqcap$ in QL).

Finally, note that none of the profiles allow unions on the right, or negations or universals on the left. This has fundamental reasons that will be explained below.

Summary The essential differences between the profiles are in their restrictions on inverse properties, and universal and existential quantifiers. One can allow the same Boolean features ($\sqcap$, $\sqcup$, $\neg$, $\top$, $\bot$) in all profiles without major effects on reasoning.

4.2 Measuring Complexity

In order to understand the limits of lightweight ontology languages, we first need to talk a bit about what we mean by *lightweight*. The question that we are generally asking is: given a certain ontology language, is there an algorithm that solves the standard reasoning tasks for this language in polynomial time? If the answer to this question is *yes*, then we can show this by specifying a suitable algorithm and showing that it solves the problem. We have done this above in various cases. However, if the answer is *no*, we cannot show this by failing to find a suitable algorithm – maybe we just did not look hard enough. Therefore, a different approach is needed to show *hardness* of a computation task. We now recall the basic ideas and introduce some notation. Readers who are in need of a more thorough introduction should consult a textbook, e.g., [41].

The general vehicle to measure the hardness of a problem on computer science is the *Turing machine*. It provides us with a computational model: a standard platform on which we can "implement" algorithms. We will use the following notation for Turing machines:

Definition 51. A (non-deterministic) *Turing machine* (TM) $\mathcal{M}$ is a tuple $\langle Q, \Sigma, \Delta, q_0 \rangle$ where

- Q is a finite set of *states*,
- Σ is a finite *alphabet* that includes a *blank* symbol $\square$,
- $\Delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{l, r\})$ is a *transition relation*, and
- $q_0 \in Q$ is the *initial state*.

A *configuration* of a TM is given by its current state $q \in Q$, the sequence of alphabet symbols that is currently written on the tape, and the current position of the read/write head on the tape. The tape is of unbounded length, but a configuration can be represented finitely by only including non-blank symbols. Initially, the tape only contains a (finite) input sequence and the TM is at the first position of the tape in state q_0.

The transition relation specifies how to change between configurations. A tuple $\langle q, \sigma, q', \sigma', r \rangle$ should be read as follows: if the machine is in state q and reads symbol σ at its current position on the tape, then the machine will change its internal state to q', write the symbol σ' to the tape, and move the read/write head to the $r$ight. In any configuration, the TM might have one, many, or no possible transitions to chose from. If it has many, the TM non-deterministically follows one of them. A TM where there is at most one transition for every state q and symbol σ is *deterministic*. If there are no possible transitions, the machine *halts*. The output of the computation then is the current content of the tape. Sometimes, we are only interested in whether the TM halts at all (accepts an input), and will ignore the output.

The Turing machine can perform a computation in a discrete number of computation steps (clock ticks), using a certain portion of the tape memory to store intermediate and final results. We can therefore measure complexity by counting the number of steps (time) or the number of memory cells (space) that a TM needs to solve a problem.

For example, a problem can be solved in (deterministic) polynomial time if there is a polynomial function f and a deterministic Turing machine $\mathcal{M}$ that solves the problem after at most $f(s)$ steps, where s is the size of the input. Such statements only make sense if the "problem" is a general class of problems (e.g., checking consistency of any $\mathcal{ALCI}$ ontology), rather than an individual problem instance (e.g., checking consistency of the ontology from Example 19). For a single problem instance (or finite set of instances), the maximal number of steps is always bounded by some constant. In other words, a measure like polynomial time complexity describes the runtime behaviour of the algorithm if the input problem instances are getting larger and larger (without end). One therefore also speaks of *asymptotic* complexity. A complexity class is defined by specifying a model of computation (e.g., deterministic or non-deterministic Turing machine) and a class of functions to asymptotically bound time or space.

This explains how we can measure algorithmic complexity and how to define complexity classes such as polynomial time. However, it does not solve the problem of showing that there cannot be a polynomial algorithm for a given problem. To address this, computational complexity theory introduces the idea of *hardness*. Intuitively speaking, a problem is *hard* for a complexity class if solving the problem would also "easily" lead to a solution of any other problem in that complexity class. To make this more precise, we need to recall some basic definitions:

– A *decision problem* is given by a set P of input sequences (those that the TM should accept). In practice, we often specify problems in some more convenient notation, knowing that they could be suitably encoded for a Turing machine. A TM *solves* a decision problem P if it accepts every input from P, and rejects every other input.
– A decision problem *belongs to* a complexity class if it is solved by a TM in the constraints of the complexity class (e.g., deterministically in polynomial time).
– A decision problem P can be *reduced to* a decision problem Q by a Turing machine $\mathcal{M}$ if, for every input string w_{in}, the machine $\mathcal{M}$ halts after computing an output string w_{out}, such that $w_{\mathsf{in}} \in P$ if and only if $w_{\mathsf{out}} \in Q$.
– A decision problem P is *hard* for a complexity class C, if every decision problem from C can be reduced to P by a deterministic TM that runs in polynomial time.[11]
– A decision problem P is *complete* for a complexity class C if P belongs to C and is hard for C.

To show that a problem P is hard, we therefore need to show that all other problems of a certain class can easily be reduced to it. To do this, it is enough to take another problem that is already known to be hard and show that this problem can be reduced to P (where the reduction is of strictly lower complexity than the given class; see footnote 11). Typical complexity classes are defined in such a way that their first hard problem is immediate. For example, for the class of non-deterministically polynomial time solvable decision problems, a hard problem is the question whether a given non-deterministic Turing machine accepts an input in polynomial time. By showing other problems to be hard, we obtain a bigger choice of problems that we can conveniently use for reduction proofs.

How can hardness be used to show that a given problem cannot be solved by any polynomial time algorithm? Ideally, we can establish hardness for a complexity class

[11] The definition of hardness through deterministic polynomial time reductions only works with complexity classes that are (expected to be) strictly harder than this. We will not consider any other classes here.

that is known to contain strictly more problems than P. For example, it is known that some problems that can be solved in exponential time (ExpTime) cannot be solved in P. Hence, no ExpTime-hard problem can admit a polynomial algorithm. In other cases, it is not known if a complexity class does really contain more problems than P. Most famously, we do not know if NP $\neq$ P. However, if a problem is hard for NP, then solving it in polynomial time would also lead to polynomial algorithms for all problems in NP. The fact that a large amount of practically important problems are hard for NP, while nobody managed to find a polynomial algorithm for any of them yet, is therefore a strong evidence that no such algorithm exists. In any case, we don't have it. Therefore, even NP-hardness can be used to show that the existence of a polynomial algorithm to solve a problem is at least very unlikely, even if not theoretically impossible.

Summary Decision problems are classified in complexity classes according to the resources a Turing machine needs to solve them. Hardness can be used to argue that no polynomial time algorithm exists (or is likely to exist) for solving a certain problem.

4.3 Unions are Hard

We have already noted that all profiles avoid unions of classes on the right-hand side of subclass inclusion axioms. In this section, we explain the reason for this: reasoning would become hard for NP and thus intractable if unions were allowed on the right. Intuitively speaking, unions introduce a kind of non-determinism that requires a reasoning algorithm to make guesses for checking ontology consistency.

The result is fairly easy to show by using a well-known NP-complete problem:

Definition 52. An instance of the *3-satisfiability problem* (3SAT) is given by a propositional logic formula of the form

$$(a_{11} \vee a_{12} \vee a_{13}) \wedge \ldots \wedge (a_{n1} \vee a_{n2} \vee a_{n3})$$

where n is a number and every a_{ij} is either a propositional letter or a negated propositional letter. The decision problem 3SAT is to decide whether the formula is satisfiable (i.e., becomes true for some assignment of truth values to propositional letters).

It is not hard to reduce 3SAT to the consistency checking problem of OWL ontologies that are allowed to use union and intersection:

Theorem 53. Consistency checking in any ontology language that allows $\sqcap$ on the left and $\sqcup$ on the right of subclass inclusion axioms, as well as $\top$ and $\bot$, is NP-hard.

Proof. Consider a propositional formula as in Definition 52. For every propositional letter p, we introduce a class name A_p, and for every disjunction of the form $(a_{i1} \vee a_{i2} \vee a_{i3})$, we construct a TBox axiom. Ignoring the order of formulae in the disjunction, there are four possible cases that we translate as follows:

$$(p_1 \vee p_2 \vee p_3) \mapsto \top \sqsubseteq \mathsf{A}_{p_1} \sqcup \mathsf{A}_{p_2} \sqcup \mathsf{A}_{p_3}$$
$$(\neg p_1 \vee p_2 \vee p_3) \mapsto \mathsf{A}_{p_1} \sqsubseteq \mathsf{A}_{p_2} \sqcup \mathsf{A}_{p_3}$$

$$(\neg p_1 \vee \neg p_2 \vee p_3) \mapsto A_{p_1} \sqcap A_{p_2} \sqsubseteq A_{p_3}$$
$$(\neg p_1 \vee \neg p_2 \vee \neg p_3) \mapsto A_{p_1} \sqcap A_{p_2} \sqcap A_{p_3} \sqsubseteq \bot$$

Let O be the resulting ontology. If O is consistent, then it has a model $\mathcal{I}$. Every model contains at least one domain element e. For every class A_p, we have either $e \in A_p^{\mathcal{I}}$ or $e \notin A_p^{\mathcal{I}}$. This defines an assignment of truth values for the original 3SAT formula: p maps to *true* if $e \in A_p^{\mathcal{I}}$, and to false otherwise. It is easy to see that this assignment makes the 3SAT formula true, since $\mathcal{I}$ satisfies every TBox axiom in O.

Conversely, every valid assignment of truth values for the 3SAT formula can be used to construct a model for O (with just one domain element). Thus, it is clear that O is consistent if and only if the original formula was satisfiable. The reduction can be computed in polynomial time. □

This shows that adding unions on the right-hand side of class inclusion axioms would make OWL EL intractable. For OWL RL, there is a minor technical difficulty since $\top$ is not allowed. It is easy to see that we can replace $\top$ with an arbitrary new class name A, and add an additional axiom $A(c)$ for some individual c. A similar reduction is then possible.

The situation is not so clear for OWL QL in its official form, where no intersections are allowed in classes on the left-hand side. Indeed, the ontology language that only allows arbitrary unions but no intersections would still allow polynomial reasoning. However, given that we need to chose between $\sqcap$ on the left and $\sqcup$ on the right, it seems clear that the former is more practical in most applications. Theorem 53 asserts that we cannot have both.

Finally, note that the result also explains why we do not allow complement $\neg$ on the left. For example, the disjunction $\neg p_1 \vee p_2 \vee p_3$ could directly be encoded as

$$A_{p_1} \sqcap \neg A_{p_2} \sqcap \neg A_{p_3} \sqsubseteq \bot$$

if this feature would be allowed. A similar but slightly more subtle encoding is possible with universal quantifiers on the left. This time, we use property names P_p to encode propositions p. The disjunction can then be expressed as

$$\exists P_{p_1}.\top \sqcap \forall P_{p_2}.\bot \sqcap \forall P_{p_3}.\bot \sqsubseteq \bot.$$

A little consideration shows that this is essentially the same encoding as for complements, since $\forall P.\bot$ means that there are no P-relations.

Summary In any ontology language that allows $\sqcap$ on the left of TBox axioms, reasoning becomes intractable (NP-hard) when allowing $\sqcup$ on the right, or $\neg$ or $\forall$ on the left.

4.4 Showing ExpTime-Hardness with Alternating Turing Machines

After settling the case of $\sqcup$, the remaining option for extending the profiles is to allow additional quantifiers in class inclusion axioms, i.e., to combine the features of two of the three profiles. It turns out that this makes reasoning significantly more difficult: the standard reasoning problems become hard for ExpTime (the class of problems solvable in exponential time on a deterministic TM) rather than just for NP (the class of problems solvable in polynomial time on a non-deterministic TM). In practice, both classes require us to use algorithms that need exponentially many computation steps in the worst case, yet there is a big qualitative difference between the two: it is known that ExpTime contains strictly more problems than P, whereas this is not clear for NP. Therefore, different approaches are necessary to show hardness for ExpTime. The approach that we will use is based on a particularly elegant characterisation of ExpTime that uses a special kind of computational model: the *alternating Turing machine*.

An alternating Turing machine (ATM) consists of the same components as the normal TM introduced in Section 4.2: a set of states Q with an initial state q_0, a tape alphabet Σ, and a transition relation that defines one, none, or many transitions that the ATM can perform in each step. The difference between ATMs and TMs is in the acceptance condition. A normal TM accepts an input if there is a (non-deterministic) choice of transitions for getting from the initial configuration to a final configuration (one in which no further transitions are possible). In other words, a configuration of a normal TM is accepting if it is either final, or there *exists* a transition that leads to an accepting configuration (defined recursively). What happens if we change this existential statement into a universal one, requiring that *all* possible transitions lead to an accepting state? We could think of this as a point where there Turing machine "forks" to create many identical copies of its current state (including the tape contents), so that each possible transition can be explored in parallel by one of the TM copies.

The special power of an ATM is that it can alternate between these two modes: in some states, it will only explore *one* possible transition non-deterministically, in other states, it will fork and explore *all* possible transitions in parallel. To control this, the set of states Q is partitioned into a set E of *existential states* and a set U of *universal states*. A configuration is *accepting* if one of the following conditions hold:

– The configuration is final, i.e., there is no possible transition.
– The ATM is in an existential state and there is a transition that leads to an accepting configuration.
– The ATM is in a universal state and all transitions lead to an accepting configuration.

Note that the second and third cases of this definition are recursive. An ATM *accepts* a given input if its initial configuration is accepting. Note that this computational model only works well for decision problems where we only want to know if the ATM accepted an input or not. ATMs do not produce any output.

A non-deterministic TM can be viewed as an ATM that has only existential states. In a sense, universal states can be used to model the dual of the acceptance condition of a non-deterministic TM. For example, to check whether a propositional formula is satisfiable, a non-deterministic TM can check if there exists one possible way of assigning truth values to propositional letters that makes the formula true. This is a classical NP problem. If, dually, we want to check if a propositional formula is unsatisfiable, then we need to verify that all possible truth assignments evaluate to false. This is a so-called co-NP problem, i.e., the dual of an NP problem. Using ATMs, we can express both types of problems in a uniform way. Moreover, we can arbitrarily combine existential and universal modes of computation, resulting in a real gain in computational power.

When we measure the resources (time and space) needed by an ATM, we consider each of its computation paths individually: even if many parallel copies of the ATM have been created during the computation, we only measure the resources used by any one of them. For example, the space (memory) needed by an ATM is the maximal size of any configuration that is considered in the computation. Likewise, the time (steps) needed by an ATM on an accepting run is the maximal number of transitions used to get from the initial to a final configuration. The kind of ATMs that we are interested in have a limited amount of memory:

Definition 54. An ATM *accepts* an input in space s if it uses at most s tape cells in any accepting configuration. A decision problem P is solved by a *polynomially space-bounded* ATM if there is a polynomial p such that an input sequence w is in P if and only if w is accepted in space $p(|w|)$, where $|w|$ is the length of w.

The special power of ATMs lies in the fact that they can solve ExpTime problems in polynomial space:

Theorem 55. The complexity class APSpace of languages accepted by polynomially space-bounded ATMs coincides with the complexity class ExpTime.

Summary Alternating Turing machines (ATMs) generalise TMs through existential and universal acceptance conditions. ATMs solve ExpTime problems in polynomial space.

4.5 Universal + Existential = Exponential

The ontology language that is obtained by combining OWL RL and OWL EL is essentially the same that is obtained by combining OWL RL and OWL QL. In either case, we obtain the sublanguage of $\mathcal{ALCI}$ that allows $\sqcap$ and $\exists$ on the left, and $\exists$ and $\forall$ on the right. In this section, we show that this already leads to ExpTime-hard reasoning problems that can certainly not be solved by a deterministic polynomial time algorithm. To see this, we "simulate" the computation of an ATM using an ontology. More precisely, given an ATM $\mathcal{M}$ and an input sequence w, we construct an ontology $\mathcal{O}_{\mathcal{M},w}$ such that $\mathcal{M}$ accepts w if and only if $\mathcal{O}_{\mathcal{M},w}$ entails a certain class inclusion axiom. In other words, we reduce ATM acceptance to subsumption checking.

How can we simulate a Turing machine in an ontology? In essence, we need to create an ontology that describes what it means for $\mathcal{M}$ to accept w. Therefore, we need to represent configurations, possible transitions between them, and the acceptance condition, using ontological axioms only. Intuitively, the domain elements of a model of the ontology will represent possible configurations of $\mathcal{M}$, encoded with the help of the following class names:

- A_q : the ATM is in state q,
- H_i : the ATM's read/write head is at position i on the storage tape,
- $\mathsf{C}_{\sigma,i}$: position i on the storage tape contains symbol σ,
- Acc : the ATM accepts this configuration.

For example, to express that c is a configuration where the ATM is in state q at position 2 of the tape, and the tape contains the sequence of letters *example*, we could use the following axiom:

$$A_q \sqcap H_2 \sqcap C_{e,0} \sqcap C_{x,1} \sqcap C_{a,2} \sqcap C_{m,3} \sqcap C_{p,4} \sqcap C_{l,5} \sqcap C_{e,6}(c)$$

In order for our approach to work, however, the ontology $O_{\mathcal{M},w}$ must be constructed in polynomial time from the input $\mathcal{M}$ and w. Therefore, the key to our encoding is that we only need a relatively small number of class names. For example, there are only a linear number of ATM states, hence there are only linearly many class names A_q. Moreover, since the ATM is polynomially space-bounded, we only need to consider a polynomial number of possible head positions i for H_i. For every position i, we need one class $C_{\sigma,i}$ for each possible alphabet symbol σ (of which there are only linearly many), so the overall number of classes $C_{\sigma,i}$ is again polynomial. This is why it is so convenient to use ATMs here: with a classical ExpTime TM it would be possible to use an exponential amount of memory, and it would not be that easy to encode this in a polynomial ontology. In order to describe possible transitions between configurations, we also introduce some property names:

- S_δ : connects two configurations if the second can be reached from the first using the transition $\delta \in \Delta$ (where δ is a tuple $\langle q, \sigma, q', \sigma', \mathit{direction}\rangle$)

Again, there are only a linearly many transitions δ, so the number of properties S_δ is small.

Now to describe the ontology $O_{\mathcal{M},w}$ in detail, consider a fixed ATM $\mathcal{M}$ with states $Q = U \cup E$, initial state q_0, tape alphabet Σ, and transition relation Δ. The sets U and E specify the universal and existential states as in Section 4.4. Moreover, $\mathcal{M}$ is polynomially space-bounded, i.e., there is a polynomial p that defines for a given input sequence w an upper bound $p(|w|)$ for the number of tape cells that will be needed.

Now for a given input word w, the ontology $O_{\mathcal{M},w}$ is defined to contain the axioms in Fig. 6. Every axiom is taken for all combinations of states, positions, alphabet symbols, and transition relations for which the conditions on the side are satisfied. Using the intuitive meaning of class and property names, it is not hard to understand each axiom:

(1) Left and right transition rules. Each of these axioms encodes a possible transition of the form $\delta = \langle q, \sigma, q', \sigma', \mathit{direction}\rangle$. The left-hand classes express that the ATM is in state q on position i, reading the symbol σ. The right-hand class then asserts that the configuration obtained by applying δ can be reached through property S_δ. The side conditions ensure that we do not accidentally move the ATM head out of the available memory area.
(2) Memory. These axioms make sure that tape cells which are not changed are not forgotten during a transition. Whenever the tape contains σ at position i but the head is at a different position j, then every successor configuration must also contain σ at position i.
(3) Final configurations. If the ATM is in a state q reading symbol σ, and there is no possible transition for this combination, then the configuration is accepting.
(4) Existential acceptance. If the ATM is in an existential state and there is any transition to an accepting configuration, then the current configuration is also accepting.

The following axioms are instantiated for all states $q, q' \in Q$, alphabet symbols $\sigma, \sigma' \in \Sigma$, tape positions $i, j \in \{0, \ldots, p(|w|) - 1\}$, and transitions $\delta \in \Delta$:

(1) Left and right transition rules

$\mathsf{A}_q \sqcap \mathsf{H}_i \sqcap \mathsf{C}_{\sigma,i} \sqsubseteq \exists \mathsf{S}_\delta.(\mathsf{A}_{q'} \sqcap \mathsf{H}_{i+1} \sqcap \mathsf{C}_{\sigma',i})$ if $\delta = \langle q, \sigma, q', \sigma', r\rangle$ and $i < p(|w|) - 1$

$\mathsf{A}_q \sqcap \mathsf{H}_i \sqcap \mathsf{C}_{\sigma,i} \sqsubseteq \exists \mathsf{S}_\delta.(\mathsf{A}_{q'} \sqcap \mathsf{H}_{i-1} \sqcap \mathsf{C}_{\sigma',i})$ if $\delta = \langle q, \sigma, q', \sigma', l\rangle$ and $i > 0$

(2) Memory

$\mathsf{H}_j \sqcap \mathsf{C}_{\sigma,i} \sqsubseteq \forall \mathsf{S}_\delta.\mathsf{C}_{\sigma,i}$ if $i \neq j$

(3) Final configurations

$\mathsf{A}_q \sqcap \mathsf{H}_i \sqcap \mathsf{C}_{\sigma,i} \sqsubseteq \mathsf{Acc}$ if there is no transition from q and σ

(4) Existential acceptance

$\mathsf{A}_q \sqcap \exists \mathsf{S}_\delta.\mathsf{Acc} \sqsubseteq \mathsf{Acc}$ if $q \in E$

(5) Universal acceptance

$\mathsf{A}_q \sqcap \mathsf{H}_i \sqcap \mathsf{C}_{\sigma,i} \sqcap \bigsqcap_{\delta \in \Delta(q,\sigma)} \exists \mathsf{S}_\delta.\mathsf{Acc} \sqsubseteq \mathsf{Acc}$ if $q \in U$ and where $\Delta(q, \sigma)$ is the set of all transitions from q and σ

Fig. 6. Knowledge base $\mathcal{O}_{\mathcal{M},w}$ simulating a polynomially space-bounded ATM

(5) Universal acceptance. If the ATM is in a universal state q reading symbol σ, and if it has an accepting successor configuration for all possible transitions, then the current configuration is also accepting. Note how we avoid the use of universal quantifiers on the left by explicitly requiring accepting successor configurations for each possible transition. This is the reason why we use special successor relations S_δ for each transition δ.

Together, these axioms ensure that domain elements of every model of the ontology $\mathcal{O}_{\mathcal{M},w}$ can be interpreted as ATM configurations that are related with the expected transition relations. This works, even though we are not very strict in enforcing that every domain element is really some valid configuration. On the one hand, we do not require that all necessary information (state, position, tape content) is specified for all elements. Indeed, to express a disjunctive statement like "in every configuration, the ATM is in one of its states" we would need some kind of class union on the right. On the other hand, we also do not require that all of the information is specified consistently (just one state and position at once, just one symbol on each tape cell). Overall, the axioms only state that, whenever a domain element can be viewed as a (partial) representation of an ATM configuration, it should also have the according successor configurations and acceptance status. This turns out to be enough to capture the behaviour of the ATM.

Now the initial configuration for the input w of the form $\sigma_0, \sigma_1, \ldots, \sigma_{|w|-1}$ is described by the following class I_w:

$$\mathsf{I}_w \coloneqq \mathsf{A}_{q_0} \sqcap \mathsf{H}_0 \sqcap \mathsf{C}_{\sigma_0,0} \sqcap \ldots \sqcap \mathsf{C}_{\sigma_{|w|-1},|w|-1} \sqcap \mathsf{C}_{\square,|w|} \sqcap \ldots \sqcap \mathsf{C}_{\square,p(|w|)-1},$$

where we also specify that all unused tape cells contain the blank symbol. We will show that checking whether the initial configuration is accepting is equivalent to checking whether $I_w \sqsubseteq \mathsf{Acc}$ follows from $\mathcal{O}_{\mathcal{M},w}$.

First we specify the relationship between elements of an interpretation that satisfies $\mathcal{O}_{\mathcal{M},w}$ and configurations of $\mathcal{M}$, which we have only sketched so far. For this, consider an interpretation $\mathcal{I}$ of $\mathcal{O}_{\mathcal{M},w}$ and an element $e \in \Delta^{\mathcal{I}}$. Let α be an ATM configuration where $\mathcal{M}$ is in state q at position i, and the tape contains the symbols $\sigma_0 \ldots \sigma_{p(|w|)-1}$. We say that e *represents* this configuration α if $e \in A_q^{\mathcal{I}}$, $e \in H_i^{\mathcal{I}}$ and $e \in C_{\sigma_j,j}^{\mathcal{I}}$ for every $j = 0, \ldots, p(|w|) - 1$. Again, observe that a single element might represent more than one configuration. We will see that this does not affect our results. If e represents a configuration, we will also say that e has state q, position i, symbol σ_j at position j etc.

Lemma 56. Consider a model $\mathcal{I}$ of $\mathcal{O}_{\mathcal{M},w}$. If some element e of $\mathcal{I}$ represents a configuration α and some transition δ is applicable to α, then e has an $S_\delta^{\mathcal{I}}$-successor that represents the (unique) result of applying δ to α.

Proof. Consider an element e, state α, and transition δ as in the claim. Then one of the axioms (1) of Fig. 6 applies, and e must also have an $S_\delta^{\mathcal{I}}$-successor. This successor represents the correct state, position, and symbol at position i of e, again by the axioms (1). By axiom (2), symbols at all other positions are also represented by all $S_\delta^{\mathcal{I}}$-successors of e. □

The next lemma shows the correctness of our ATM simulation.

Lemma 57. The input sequence w is accepted by $\mathcal{M}$ if and only if $I_w \sqsubseteq \mathsf{Acc}$ is a consequence of $\mathcal{O}_{\mathcal{M},w}$.

Proof. Consider an arbitrary interpretation $\mathcal{I}$ that satisfies $\mathcal{O}_{\mathcal{M},w}$. We first show that, if any element e of $\mathcal{I}$ represents an accepting configuration α, then $e \in \mathsf{Acc}^{\mathcal{I}}$. We use an inductive argument along the recursive definition of acceptance. As a base case, assume that α is a final configuration (without transitions). Then axiom (3) applies and we find that $e \in \mathsf{Acc}^{\mathcal{I}}$ as required.

For the induction step, first assume that α is an existential configuration. Then there is some accepting δ-successor configuration α' of α. By Lemma 56, there is an $S_\delta^{\mathcal{I}}$-successor e' of e that represents α', and we find $e' \in \mathsf{Acc}^{\mathcal{I}}$ by the induction hypothesis. Hence axiom (4) applies and we conclude $e \in \mathsf{Acc}^{\mathcal{I}}$.

As the remaining case, assume that α is a universal configuration. Then all successors of α are accepting, too. By Lemma 56, for any δ-successor configuration α' of α, there is a corresponding $S_\delta^{\mathcal{I}}$-successor e' of e. By the induction hypothesis for α', we find $e' \in \mathsf{Acc}^{\mathcal{I}}$. Since this holds for all δ-successors of α, axiom (5) implies $e \in \mathsf{Acc}^{\mathcal{I}}$. This finishes the induction.

Since all elements in $I_w^{\mathcal{I}}$ represent the initial configuration of the ATM, this shows that $I_w^{\mathcal{I}} \subseteq \mathsf{Acc}^{\mathcal{I}}$ whenever the initial configuration is accepting.

It remains to show the converse: if the initial configuration is not accepting, there is some interpretation $\mathcal{I}$ such that $I_w^{\mathcal{I}} \nsubseteq \mathsf{Acc}^{\mathcal{I}}$. To this end, we define a canonical interpretation $\mathcal{J}$ of $\mathcal{O}_{\mathcal{M},w}$ as follows. The domain of $\mathcal{J}$ is the set of all configurations of $\mathcal{M}$ that encode a tape of length $p(|w|)$. The interpretations for the classes A_q, H_i, and $C_{\sigma,i}$ are defined as expected so that every configuration represents itself but no other configuration. Especially, $I_w^{\mathcal{J}}$ is the singleton set containing the initial configuration. Given two configurations α and α', and a transition δ, we define $\langle \alpha, \alpha' \rangle \in S_\delta^{\mathcal{J}}$ if and only if there is a transition δ from α to α'. $Acc^{\mathcal{J}}$ is defined to be the set of accepting configurations.

By checking the individual axioms of Fig. 6, it is easy to see that $\mathcal{J}$ satisfies $O_{\mathcal{M},w}$. Now if the initial configuration is not accepting, $I_w^{\mathcal{J}} \not\subseteq Acc^{\mathcal{J}}$ by construction. Thus $\mathcal{J}$ is a counterexample for $I_w \sqsubseteq Acc$, which thus is not a logical consequence. □

Theorem 58. The standard reasoning problems are ExpTime-hard for any ontology language that allows $\forall$ on the right-hand side of subclass expressions, and $\sqcap$ and $\exists$ (with arbitrary filler classes) on the left and right. In particular, this is the case for the combination of $\mathcal{EL}$*tiny* and $\mathcal{RL}$*tiny*, and for the combination of $\mathcal{RL}$*tiny* and $\mathcal{QL}$*tiny*.

Proof. Lemma 57 shows that the acceptance problem for polynomially space-bounded ATMs can be reduced to checking class subsumption in $O_{\mathcal{M},w}$. The other standard reasoning problems can be reduced to satisfiability checking as discussed on Section 2.2. The reduction is polynomially bounded due to the restricted number of axioms: there are at most $p(|w|)\times|\Delta|$ axioms of type (1), $p(|w|)^2\times|\Sigma|\times|\Delta|$ axioms of type (2), $|Q|\times p(|w|)\times|\Sigma|$ axioms of type (3), $|Q| \times |\Delta|$ axioms of type (4), and $|Q| \times p(|w|) \times |\Sigma|$ axioms of type (5). The claim then follows from Theorem 55. □

Summary Reasoning becomes ExpTime-hard when combining OWL RL with OWL EL or OWL QL, which can be shown by simulating a polynomially space-bounded ATM.

4.6 OWL EL + OWL QL = ExpTime

It remains to investigate the combination of OWL EL and OWL QL. On the first glance, the features of the two profiles may seem very similar, and one could thus hope that the combination of both languages would not lead to major difficulties. Unfortunately, this case leads to the same exponential complexity that we already got for the combinations with OWL RL in the previous section. Thankfully it is at least easy to prove this now.

It turns out that we can largely re-use the proof of Section 4.5. Looking at the axioms in Fig. 6, we can see that the memory axioms (2) are the only ones that we cannot readily express using the features of $\mathcal{EL}$*tiny* alone. Of course, (2) is not in $\mathcal{QL}$*tiny* either, but we can use inverse properties to write it in a different form:

$$\exists S_\delta^-.(H_j \sqcap C_{\sigma,i}) \sqsubseteq C_{\sigma,i} \qquad \text{if } i \neq j$$

It is easy to see that this axiom is equivalent to (2). Therefore, all previous proofs can be applied using this alternative axiom, and we obtain the following result:

Theorem 59. The standard reasoning problems are ExpTime-hard for any ontology language that allows inverse properties, $\sqcap$, and $\exists$ (with arbitrary filler classes). In particular, this is the case for the combination of $\mathcal{EL}$*tiny* and $\mathcal{QL}$*tiny*.

Summary OWL EL has all features of OWL QL other than inverse properties. Adding them makes all standard reasoning tasks ExpTime-hard.

5 Advanced Modelling Features

The little ontology languages $\mathcal{EL}\mathit{tiny}$, $\mathcal{RL}\mathit{tiny}$, and $\mathcal{QL}\mathit{tiny}$ have served us well for illustrating the most essential characteristics of the three OWL profiles. In this section, we will complete the picture by mentioning the most important additional features that can be found in the OWL profiles. Only a few of these have a real impact on computation, and all of the reasoning methods introduced in Section 3.1 can be adopted to the larger languages.

Property Hierarchies Just like in the case of classes, we can also specify that a property is more specific than another one. For example, we could state

$$\mathit{hasMother} \sqsubseteq \mathit{hasParent}$$

to say that motherhood is a special form of parenthood. In OWL this is encoded using SubObjectPropertyOf, which is supported by all profiles. Property hierarchies can also be used to declare two properties to be equivalent by stating that they mutually include each other.

Property Chains This generalisation of property hierarchies allows us to state that the combination of two properties leads to a new relationship. A typical example is

$$\mathit{hasParent} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle}$$

which states that the brother of someone's parent is one's uncle. In OWL this is encoded using SubObjectPropertyChain, and some restrictions apply regarding the use of this feature (which, by the way, are unnecessary in the profiles). Only OWL EL and OWL RL support property chains, while OWL QL disallows them since they would make query rewriting in the sense of Section 3.8 impossible. To stay polynomial, OWL EL needs to impose some further restrictions regarding the interplay of property range axioms $\top \sqsubseteq \forall P.C$ and property chains. A special case of property chain can be used to express *transitivity*, as in the following example:

$$\mathit{hasAncestor} \circ \mathit{hasAncestor} \sqsubseteq \mathit{hasAncestor}$$

Equality OWL allows us to express that two individual names refer to the same thing, e.g., to declare two different names for Tweety:

$$\mathit{tweety} \approx \mathit{tweetyBird}$$

In OWL this is encoded using SameIndividual, which is supported in OWL EL and OWL RL but not in OWL QL. One can also say the opposite, i.e., that two individuals cannot be the same. This is supported by all profiles, but it is uninteresting in OWL QL, where there is no reason why two individuals should ever be equal.

Nominals A special form of class expressions allows us to specify classes with exactly one element, called *nominals* in description logics. For example,

$$\exists \mathit{livesIn}.\{\mathit{europe}\} \sqsubseteq \mathit{European}$$

states that everybody who lives in (the individual) Europe belongs to the class of Europeans. Nominals can be very powerful modelling constructs that can also be used to encode ABox statements, equality, and inequality. In OWL, nominals are expressed using ObjectOneOf and (in constructions as in the above example) ObjectHasValue. OWL RL and OWL EL support these features, while OWL QL does not allow them.

Local Reflexivity: Self This feature allows us to speak about things that are related to themselves with some property. For example,

$$\mathit{CEO} \sqsubseteq \exists \mathit{isSupervisorOf}.\mathsf{Self}$$

states that every CEO is her own supervisor. This is expressed in OWL using ObjectHasSelf. This is only allowed in OWL EL, although there is no technical reason to exclude it from OWL RL.[12]

Datatypes OWL supports a variety of datatypes such as numbers, strings, and booleans. They can be used as values to specific properties, called *data properties*. In contrast, the properties we have used so far are called *object properties* (hence the word Object in most OWL language features that we have seen). Many kinds of statements that we have seen before can also be made with data properties:

$\mathit{hasName}(\mathit{tweety}, \texttt{"Tweety Bird"})$	ABox axiom that assigns a string value
$\mathit{Person} \sqsubseteq \exists \mathit{hasName}.\texttt{String}$	Every person has a string name
$\mathit{firstName} \sqsubseteq \mathit{hasName}$	Every first name is also a name

However, data properties always relate individuals to data values, hence they cannot be chained or inverted. All profiles of OWL support data properties. The main difference is in the set of available datatypes. The profiles OWL EL and OWL QL that allow existential data property restrictions on the right-hand side of class inclusions support a more limited set of datatypes than OWL RL. As in the case of class unions, these restrictions could be lifted on the left-hand side of class inclusions. For OWL QL, it is noteworthy that existential restrictions on data properties are not restricted when occurring on the left-hand side of class expressions. The difficulties explained in Example 43 do not occur in this case.

Keys It is possible to state that the individuals of some class are uniquely identified by certain *keys*. This means, whenever two individuals have the same property values for a given list of properties, it is inferred that they are equal. There is no dedicated syntax for this in description logics. An example in OWL notation is:

$$\mathsf{HasKey}(\ \mathit{Person}\ \mathit{bornIn}\ \mathit{hasName}\ \mathit{birthday}\ )$$

[12] The author is not aware of any investigation of this feature in OWL QL and conjectures that Self could be included there as well.

which states that two persons that are born in the same place, and have the same name and birthday are necessarily the same. Keys are only applied to elements that are denoted by an individual name (not to elements that have been inferred to exist without a name being known for them; see Example 35). Keys are available in OWL EL and OWL RL, but not in OWL QL.

Syntactic Sugar Many further features in OWL are merely syntactic abbreviations for statements that can also be expressed with other features. For example, there are special constructs for defining disjoint classes (see Remark 7), and transitive and symmetric properties. As a general rule, these abbreviations are available in any profile where one could also express the feature indirectly.

Summary All profiles also support datatypes and property hierarchies. OWL EL and OWL RL further support equality, keys, nominals, property chains, and Self (EL only).

6 Summary and Further Reading

Lightweight ontology languages have become a highly relevant in many applications, and the standardisation of the OWL 2 profiles EL, RL, and QL has supported their practical adoption.

Each profile allows for highly efficient polynomial time reasoning algorithms. For OWL RL and OWL EL, these are typically founded on rule-based saturation approaches where logical consequences are computed in a bottom-up fashion. For OWL QL, reasoning is typically implemented through query rewriting algorithms, which generate a set of queries that can be evaluated over the ABox. This approach has the advantage that reasoning complexity mainly depends on the TBox, but it leads to a comparatively restricted set of expressive features.

To the best of our knowledge, this is the first text that treats all three OWL profiles and the related reasoning methods. A number of sources are available for further reading. The foremost reference on the OWL 2 Profiles is the W3C standard [34], and in particular the profiles document [32]. A step-by-step introduction to all features of OWL is found in the *OWL Primer* [13]. For a comprehensive treatment of OWL, RDF, and related technologies, we recommend an introductory textbook [14].

For more information on description logics, the *DL Primer* provides a gentle first exposition [28]. More detailed introductions can be found in previous lecture notes of the Reasoning Web Summer School: Rudolph provides a detailed discussion of DL semantics and modelling [38], Baader gives a general overview with extended historical notes [2], and Sattler focusses on tableau-based reasoning methods [39]. An introduction to conjunctive query answering in description logics is given by Ortiz and Simkus in another chapter of these lecture notes [33]. A basic reference for advanced topics in DL research is the *Description Logic Handbook* [5].

More information on OWL EL and its implementation can be found in a number of research papers. Basic investigations on the related description logic $\mathcal{EL}^{++}$ have first been made by Baader, Brandt, and Lutz [3], who also introduced the extension with property ranges that is used in OWL EL [4]. Implementation aspects have been discussed in detail in a series of recent works related to the OWL EL reasoner ELK [18,19,21]. A discussion of the engineering aspects of implementing such a saturation-based system can be found in a technical report [20]. Other dedicated OWL EL reasoners include CEL [6], Snorocket [29], and jCEL [31]. The general-purpose OWL reasoner Pellet uses special algorithms if the input ontology belongs to a certain subset of OWL EL, allowing it to classify some large OWL EL ontologies that would otherwise not be supported. Conjunctive query answering in OWL EL tends to be harder than in OWL QL and OWL RL, and is intractable even under additional restrictions [26]. Efficient solutions were proposed for sublanguages of OWL EL [30,24].

OWL RL is often studied as an extension of the Semantic Web data representation standard RDF [22] with additional inference rules. Relevant sublanguages that cover only some features of OWL RL include RDF Schema [7] and pD* (a.k.a. OWL-Horst) [17]. The OWL standard includes a set of inference rules that are formulated in terms of the RDF encoding of OWL [32]. In contrast to OWL EL, the main focus of many OWL RL implementations is on large datasets that need to be managed in a database system. Noteworthy RDF database systems which support a certain amount of OWL RL reasoning include AllegroGraph, Jena, OpenLink Virtuoso, Oracle 11g, OWLIM, and Sesame. Reasoning is typically implemented with saturation-based calculi, sometimes based on a configurable set of inference rules. Virtuoso can also be configured to use query rewriting for some tasks. In general, conjunctive query answering is easy in OWL RL, since the saturated ontology can simply be queried like an extended database. Academic research in OWL RL has strongly focussed on aspects of scalability and distribution; examples include [43,16,25,15,44,42]. There have also been works on data quality and cleaning in the context of reasoning, where noisy data can have stronger effects [15].

OWL QL was originally inspired by the so-called DL-Lite family of description logics [8]. The current W3C standard is closely related to the logic DL-Lite$_{\mathcal{R}}$, later called DL-Lite$^{\mathcal{R}}_{core}$ [1]. Query rewriting can often lead to a huge number of ABox queries, and various suggestions have been made to improve this [24,12,23,37]. The only freely available implementation for OWL QL appears to be Owlgres.[13] Another implementation, QuOnto,[14] is available to registered users. Some query rewriting approaches in OWL RL systems can usually also be viewed as OWL QL implementations.

The study of computational complexity is an important area of investigation in knowledge representation and reasoning, and it is often used to guide the design of ontology languages. A good textbook introduction to the field is provided by Sipser [41], a more extensive treatment is given by Papadimitriou [35]. Alternating Turing Machines were originally introduced in [9]. The proof of ExpTime-hardness for reasoning in combinations of OWL profiles that is given in Section 4.5 has first been formulated in [27], where additional related results can be found. In particular, it is shown there that the extension of OWL QL with universal quantifiers on the right, but without allowing general existential quantifiers on the left, leads to a PSpace-complete ontology language.

Most of our results can readily be applied to reasoning under the RDF-Based Semantics as well. Following Theorem 17, all of our reasoning methods are also sound

[13] http://pellet.owldl.com/owlgres
[14] http://www.dis.uniroma1.it/quonto/

(but not complete) in this case. Our use of description logic syntax was mainly for simplifying the presentation, and the inference rules in our calculi can easily be translated to OWL syntax along the lines of Table 1. For the case of OWL QL, one should also define the meaning of conjunctive query answering under RDF-Based Semantics first, which is closely related to the semantics of SPARQL queries under RDF-Based Semantics [10]. This is also part of the upcoming W3C SPARQL 1.1 *entailment regime* for OWL RDF-Based Semantics [11].

Acknowledgements The author would like to thank Yavor Nenov and an anonymous reviewer for their helpful comments. Work on this chapter was supported by the EU FP7 project SEALS and by the EPSRC projects ConDOR and ExODA.

Errata, if any, will be published at http://korrekt.org/page/OWL_2_Profiles. Feedback can be sent to the author.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. of Artificial Intelligence Research 36, 1–69 (2098)
2. Baader, F.: Description logics. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.C., Schmidt, R.A. (eds.) Reasoning Web. Semantic Technologies for Information Systems – 5th International Summer School, 2009, LNCS, vol. 5689, pp. 1–39. Springer (2009)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L., Saffiotti, A. (eds.) Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). pp. 364–369. Professional Book Center (2005)
4. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Clark, K.G., Patel-Schneider, P.F. (eds.) Proc. OWLED 2008 DC Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 496. CEUR-WS.org (2008)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
6. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—a polynomial-time reasoner for life science ontologies. In: Furbach, U., Shankar, N. (eds.) Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). LNCS, vol. 4130, pp. 287–291. Springer (2006)
7. Brickley, D., Guha, R.V. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (10 February 2004), available at http://www.w3.org/TR/rdf-schema/
8. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning 39(3), 385–429 (2007)
9. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. of the ACM 28(1), 114–133 (1981)
10. Glimm, B., Krötzsch, M.: SPARQL beyond subgraph matching. In: Patel-Schneider et al. [36], pp. 241–256
11. Glimm, B., Ogbuji, C. (eds.): SPARQL 1.1 Entailment Regimes. W3C Working Draft (05 January 2012), available at http://www.w3.org/TR/sparql11-entailment/
12. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: Abiteboul, S., Böhm, K., Koch, C., Tan, K.L. (eds.) Proc. 27th Int. Conf. on Data Engineering (ICDE'11). pp. 2–13. IEEE Computer Society (2011)

13. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language: Primer. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-primer/
14. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
15. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the Web. Int. J. of Semantic Web Inf. Syst. 5(2), 49–90 (2009)
16. Hogan, A., Pan, J.Z., Polleres, A., Decker, S.: SAOR: template rule optimisations for distributed reasoning over 1 billion linked data triples. In: Patel-Schneider et al. [36], pp. 337–353
17. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. of Web Semantics 3(2–3), 79–115 (2005)
18. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) Proceedings of the 10th International Semantic Web Conference (ISWC'11). LNCS, vol. 7032. Springer (2011)
19. Kazakov, Y., Krötzsch, M., Simančík, F.: Unchain my $\mathcal{EL}$ reasoner. In: Proceedings of the 23rd International Workshop on Description Logics (DL'10). CEUR Workshop Proceedings, vol. 745. CEUR-WS.org (2011)
20. Kazakov, Y., Krötzsch, M., Simančík, F.: ELK: a reasoner for OWL EL ontologies. Tech. rep. (2012), available from http://code.google.com/p/elk-reasoner/wiki/Publications
21. Kazakov, Y., Krötzsch, M., Simančík, F.: Practical reasoning with nominals in the $\mathcal{EL}$ family of description logics. In: Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12) (2012), to appear, available from http://code.google.com/p/elk-reasoner/wiki/Publications
22. Klyne, G., Carroll, J.J. (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (10 February 2004), available at http://www.w3.org/TR/rdf-concepts/
23. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in DL-Lite. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10). pp. 247–257. AAAI Press (2010)
24. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Walsh, T. (ed.) Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11). pp. 2656–2661. AAAI Press/IJCAI (2011)
25. Kotoulas, S., Oren, E., van Harmelen, F.: Mind the data skew: distributed inferencing by speeddating in elastic regions. In: Proc. 19th Int. Conf. on World Wide Web (WWW'10). pp. 531–540. WWW'10, ACM (2010)
26. Krötzsch, M., Rudolph, S., Hitzler, P.: Conjunctive queries for a tractable fragment of OWL 1.1. In: Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) Proc. 6th Int. Semantic Web Conf. (ISWC'07). LNCS, vol. 4825, pp. 310–323. Springer (2007)
27. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexities of Horn description logics. ACM Trans. Comp. Log. (2012), to appear; preprint available at http://tocl.acm.org/accepted.html
28. Krötzsch, M., Simančík, F., Horrocks, I.: A description logic primer. CoRR abs/1201.4089 (2012)
29. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Taylor, K., Meyer, T., Orgun, M. (eds.) Proc. 6th Australasian Ontology Workshop (IAOA'10). Conferences in Research and Practice in Information Technology, vol. 122, pp. 45–49. Australian Computer Society Inc. (2010)

30. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In: Boutilier, C. (ed.) Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09). pp. 2070–2075. IJCAI (2009)
31. Mendez, J., Ecke, A., Turhan, A.Y.: Implementing completion-based inferences for the $\mathcal{EL}$-family. In: Rosati, R., Rudolph, S., Zakharyaschev, M. (eds.) Proceedings of the international Description Logics workshop. vol. 745. CEUR (2011)
32. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-profiles/
33. Ortiz, M., Simkus, M.: Reasoning and query answering in description logics. In: Reasoning Web – 8th International Summer School 2012. LNCS, Springer (2012), to appear
34. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-overview/
35. Papadimitriou, C.H.: Computational Complexity. Addison Wesley (1994)
36. Patel-Schneider, P.F., Pan, Y., Glimm, B., Hitzler, P., Mika, P., Pan, J., Horrocks, I. (eds.): Proc. 9th Int. Semantic Web Conf. (ISWC'10), LNCS, vol. 6496. Springer (2010)
37. Pérez-Urbina, H., Motik, B., Horrocks, I.: A comparison of query rewriting techniques for DL-lite. In: Cuenca Grau, B., Horrocks, I., Motik, B., Sattler, U. (eds.) Proc. 22nd Int. Workshop on Description Logics (DL'09). CEUR Workshop Proceedings, vol. 477. CEUR-WS.org (2009)
38. Rudolph, S.: Foundations of description logics. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P.F. (eds.) Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, LNCS, vol. 6848, pp. 76–136. Springer (2011)
39. Sattler, U.: Reasoning in description logics: Basics, extensions, and relatives. In: Antoniou, G., Aßmann, U., Baroglio, C., Decker, S., Henze, N., Patranjan, P.L., Tolksdorf, R. (eds.) Reasoning Web – 3rd International Summer School, 2007, LNCS, vol. 4636, pp. 154–182. Springer (2007)
40. Schneider, M. (ed.): OWL 2 Web Ontology Language: RDF-Based Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-rdf-based-semantics/
41. Sipser, M.: Introduction to the Theory of Computation. Thomson Course Technology, international edition of second edn. (2005)
42. Soma, R., Prasanna, V.K.: Parallel inferencing for OWL knowledge bases. In: Proc. Int. Conf. on Parallel Processing (ICPP'08). pp. 75–82. IEEE Computer Society (2008)
43. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.: WebPIE: a Web-scale parallel inference engine using MapReduce. J. of Web Semantics (2011), in press, accepted manuscript, preprint available at http://www.cs.vu.nl/~frankh/postscript/JWS11.pdf
44. Weaver, J., Hendler, J.A.: Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) Proc. 8th Int. Semantic Web Conf. (ISWC'09). LNCS, vol. 5823, pp. 682–697. Springer (2009)