Chapter 1

Linked Data & the Semantic Web Standards

Aidan Hogan

Digital Enterprise Research Institute, National Univerity of Ireland, Galway

1.1 Introduction

On the traditional World Wide Web we all know and love, machines are used as brokers of content: they store, organize, request, route, transmit, receive and display content encapsulated as documents. In order for machines to process the content of documents automatically—for whatever purpose—they primarily require two things: machine-readable *structure* and *semantics*. Unfortunately, despite various advancements in the area of Natural Language Processing (NLP) down through the decades, modern computers still struggle to meaningfully process the idiosyncratic structure and semantics of natural language due to ambiguities present in grammar, coreference and word-sense.

Hence, machines require a more "formal" notion of structure and semantics using unambiguous grammar, referencing and vocabulary.

As such, various standards (both de facto and de jure) have emerged to partially STRUCTURE the Web's content using agreed-upon formal syntaxes and data-models. The current structure of the Web's content is predominantly based around the idea of *markup* whereby the different elemental parts of the content in a document are delimited by use of syntactic conventions, including matching start tags and end tags (e.g., `<title>Title of Document</title>`), nested elements, attributes, and so forth. The eXtensible Markup Language (XML) provides a generic standard for markup-style languages, allowing machines to parse XML content into a data model consisting of an ordered tree of typed strings. Other non-markup-based methods for structuring content have also become common. For example, Comma Separated Values (CSV) provides a simple syntax that allows machines to parse content into tabular (or even relational) data-structures. Recently, JavaScript Object Notation (JSON) has seen growth in adoption, providing syntax to represent content that can be parsed into nested complex objects and associative arrays.

However, as far as a machine is concerned, having formally structured content is only half the battle. Without some SEMANTICS (aka. meaning) for at least some parts of the content, machines would not be able to do much more than split the content up by its delimiters and load its structure. Much of the semantics that powers the current Web is based on consensus collected in standards (e.g., RFCs, W3C, etc.) for software developers and content creators to read and follow. The HyperText Markup Language (HTML) standard is perhaps the most prominent such example, providing a markup-based vocabulary that allows to state how a document should be rendered in a browser; for example, the `<title>` tag is used by publishers to denote the title of the document, which will then be predictably displayed by a Web browser in its top tool-bar (or tab-bar). The agreed-upon semantics for the HTML vocabulary of tags and elements, then, lie in the annotation of a HTML document for consistent rendering purposes. Other markup-based specifications on the Web (such as Rich Site Summary (RSS)) promote an agreed-upon meaning for a set of terms that fulfill the needs of specific other applications (in the case of RSS, providing details of site updates to a feed reader).

Importantly, for agreed-upon vocabularies such as HTML or RSS, the addition (or renaming/removal) of terms from the vocabulary requires a new standard, and eventually new applications to interpret the new terms accordingly: the semantics of relevant terms are enumerated in human-readable documentation and hard-coded in (often imperative) programs. Although standards such as XML Schema (XSD) can be used to assign some machine-readable semantics to XML content—such as what are the legal and/or required children of an element or attribute (e.g., an element `employee` should have a `staffID` attribute), or simple typing of elements and text values—such semantics are limited to defining constraints that define the notion of a "valid document" for some purpose, or for parsing datatypes such as integers, booleans and dates. In other words, XSD is more concerned with machine validation rather than machine readability. Furthermore, terms often serve a singular purpose within the context of a given application or a given schema: as an example, there is a `<title>` tag in both HTML and RSS, but how they should be interpreted differs significantly for the respective consumer applications.

So where does this leave the Web?

Consider a bunch of cooking enthusiasts who want to start sharing personal recipes with each other over the Web. Each participant will want to search over all recipes to find things like: "*citrus-free desserts*" or "*winter soups made from root vegetables*" or "*wine-based gravy for beef*" or "*barbeque*" and so forth. Some of the enthusiasts create a new site and invite users to enter recipes in structured (HTML) forms, allowing to state what ingredients are needed, in what quantities and units, what steps are required for preparation, and in what order. The recipes are stored in inverted keyword indexes and structured relational databases to allow for searching and querying over later. As the site's content grows, a tag-based system is created to allow users to fill in commonly searched terms not mentioned in the recipe text, like `bbq`, `gravy`, `vegan` and so forth. Users can comment on recipes to give their experience and ratings.

After all of the hard work, the users of the site are quite happy with the functionality. Users can search for content by keyword, by rating, or using faceted browsing over the ingredients of different recipes. However, some users still find it difficult to find the recipe they want. For example, Sally is allergic to citrus and although tags exist for common allergies, there are no tags for citrus. Thus, Sally has to go through each individual dessert recipe to ensure that the ingredient list does not contain lemons, oranges, limes, grapefruit, tangerines and other agrumes, or processed ingredients that themselves contain agrumes. Another user, Fred, has his eye on a recipe for Black risotto after enjoying it on holiday. Preferably, the recipe uses fresh cuttlefish, but if that is not available, whole squid can be used instead. Both of these are obscure ingredients and Fred is unsure where to find either of them in his local area. He searches through a variety of online shopping sites for local supermarkets and eventually finds fresh squid but is still unsure whether or not cuttlefish is available close-by.

Later, the maintainers of the cooking site decide to merge with another site that contains recipes for cocktails. There is much overlap between both sites in terms of the structure of input forms, ingredients used, preparation details, tags, and so forth. The maintainers of both sites decide to extend the cooking site and prepare a site-dump of the cocktail site to integrate with the cooking database. However, aside from all of the manual effort required in manually mapping and restructuring the content of the cocktail corpus, there are further alignment problems. Recipes on the cooking site are expected to have a preparation time, which is missing from the cocktail site; the cocktail site has alternative names for some ingredients, such as "cantaloupe" instead of "melon"; the cocktail recipes have no tags; and so forth. The maintainers eventually have to heavily adapt their database design to accomodate the incoming data, and hack together some imperative scripts to align terms and to seed common tags for cocktails like `non-alcoholic`, `vegan`, etc., based on ingredient lists, extending them manually.

Although this example admittedly takes some liberties, it serves to illustrate some of the shortcomings of the current Web. The advent of Web 2.0 technologies has blurred the line between users and publishers: the Web now contains a lot of user-generated content, be it primary content such as recipes, or secondary content in the form of comments, ratings, lists, tags, etc. However, content is heavily fragmented across different sites—even where there is a high degree of overlap across that content—with only a coarse layer of hyperlinks bridging individual sites. Content is often created in the context of a given site for the functionality of that site: though content may often be of general interest, it is often created with a singular purpose (e.g., manually tagging recipes containing lemons with `citrus`). As a result, content becomes locked into a site, due to some combination of licensing or technical issues, or simply because the content is not stated in a reusable way. Because so much of the content on the Web is not directly reusable, there are then high levels of redundancy in the manual creation of factual content across different sites (e.g., tagging lemon cocktails again with `citrus`). Similarly, content gained through one site cannot be used to automatically interact with another site (such as to search nearby shops for ingredients of recipes).

And so, in an effort to address these shortcomings, the primary goal of the "Semantic Web" is to make more of the Web's content available in a machine-readable format such that it can be reused for (m)any purpose(s), such that it can be automatically combined and integrated with other machine-readable content, and such that machines can (to some known extent) interpret and automatically act upon that content. For this envisaged Semantic Web, you would only need to say that "*all lemons are citrus fruits*" once: so long as you said it the right way—on the Web, using an globally agreed-upon identifier for lemon, described using an agreed-upon data-model, formalizing the claim using an agreed-upon vocabulary with well-defined meaning—the machines could do the rest.

This chapter continues by first outlining the original vision of the Semantic Web and the core components and technologies deemed necessary to make it a reality. Thereafter, we discuss the various core Semantic Web languages that have been standardized in recent years and that comprise the heart of the modern Semantic Web. Finally, we discuss *Linked Data*: a set of best-practices on how to identify Semantic Web resources and how to organize and interlink Semantic Web data published in a decentralized manner on the Web.

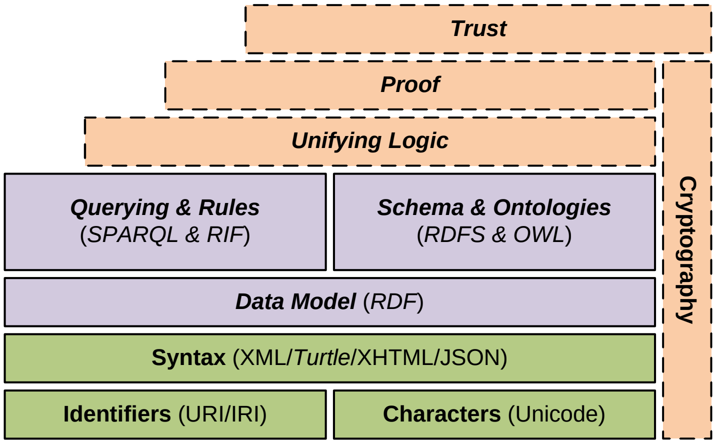

Trust
Proof
Unifying Logic
Querying & Rules (SPARQL & RIF)
Schema & Ontologies (RDFS & OWL)
Data Model (RDF)
Syntax (XML/Turtle/XHTML/JSON)
Identifiers (URI/IRI)
Characters (Unicode)
Cryptography

FIGURE 1.1: Semantic Web Stack (aka. Semantic Web Layer Cake)

1.2 Semantic Web

On a high-level, the Semantic Web can be conceptualized as an extension of the current Web so as to enable the creation, sharing and intelligent re-use of (deeply) machine-readable content on the Web. This idea of the Semantic Web is almost as old as the Web itself, and the *roots* of the Semantic Web are, of course, much older than the Web. However, two major milestones for the inception of the modern notion of the Semantic Web were the original W3C recommendation of the first Resource Description Framework (RDF) standard in February 1999 [39] outlining the core data model (described in detail later in Section 1.3), and the 2001 publication of Berners-Lee et al.'s seminal paper where the authors outlined their vision for the Semantic Web [9].

Traditionally, the technical blue-prints for building the Semantic Web from the ground up have often been represented through various incarnations of the high-level "Semantic Web Stack" (aka. "Semantic Web Layer Cake") originally conceived by Berners-Lee; yet another such incarnation is illustrated in Figure 1.1. Each layer of the stack represents a technical "piece of the puzzle" needed to realize the vision of the Semantic Web. Some parts of the puzzle already exist and can be re-used. However, much of the stack necessarily needs novel techniques; these parts are italicized in Figure 1.1.

The lower levels of the stack relate to foundational elements of the Semantic Web that are in-common with the Web itself:

Characters: Like the current Web and various other software applications, the Semantic Web requires some standard to map from binary streams and storage to textual information. For this, the Semantic Web relies on the standard Unicode character-set.

Identifiers: If the Semantic Web is about describing *things*—be they conceptual or concrete—in a machine-readable manner, these things will need globally agreed-upon identifiers. The natural choice for identifiers is thus to use the Uniform Resource Identifier (URI) specification, which is already used on the Web to identify documents (or more accurately, *representations*). Newer Semantic Web standards have also started to adopt the Internationalized Resource Identifier (IRI) specification: a generalization of URIs to support the broader Unicode standard.

Syntax: To allow machines to automatically parse content into its elementary constituents, the Semantic Web requires syntaxes with formally defined grammars. For this, existing generic syntaxes such as XML and JSON can be used. Though the use of existing syntaxes allows for using legacy tools, custom syntaxes have also been created to encode Semantic Web data using terse and intuitive grammars; these novel syntaxes are all a derivative of the Terse RDF Triple Language (Turtle) syntax.[1]

Above the **Syntax** layer lies the beating heart of the Semantic Web:

Data Model: In order for machines to exchange machine-readable data in a generic fashion, they need to agree upon a common data-model under which to structure content. This data-model should be generic enough to provide a canonical representation (without idiosyncrasies) for arbitrary content irrespective of its domain or nature or syntax, and to enable processing this content using standard off-the-shelf technologies. The core data-model elected for use on the Semantic Web is RDF, which can be serialized using one or more of the aforementioned syntaxes.

Schema & Ontologies: While the RDF data-model brings a universal structure to content, it does not bring (much) semantics or meaning to content. Thus, the Semantic Web requires formal languages with which to make claims about things described in RDF content. These formal languages offer a meta-vocabulary with well-defined semantics that can be used in combination with the RDF data-model to define schemata and ontologies. The core languages offered as part of the current Semantic Web standards are the RDF Schema (RDFS) and Web Ontology Language (OWL) standards.

Querying & Rules: Ultimately, content described in RDF needs to be processed by querying and rule-based systems that allow for specifying conjunctive conditions and query patterns. The results of conjunctive

[1] Turtle is itself inspired by Notation3 (N3). However, N3 goes beyond RDF and should not be considered an RDF syntax. Turtle can be loosely speaking the intersection of RDF and N3.

queries and rules can be used to extract pertinent elements of RDF content, to generate results for a user interface, to infer novel RDF data based on premises formed by existing content, to specify constraints that an RDF dataset should conform to, or to define triggers that perform actions when the RDF data meets certain conditions. The current querying standard for the Semantic Web is the SPARQL Protocol and RDF Query Language (SPARQL), which provides a mature, feature-rich query language for RDF content. The current standard for rules on the Semantic Web is the Rule Interchange Format (RIF), which captures the expressivity of various existing rule-based languages and offers a powerful library of built-in functions.

This chapter will primarily focus on the layers and standards enumerated above. This book will primarily focus on the support for **Querying** in the context of the RDF **Data Model** layer.

At the top and side of the stack in Figure 1.1 are a number of layers drawn with dashed lines. Although proposals to realize these layers have been made in the research literature, mature standards and tooling have yet to emerge. These are speculative areas of the Semantic Web, and that is reflected in the following discussion:

Unifying Logic: Lower down in the stack lie the query languages, rule primitives and ontological standards that are compatible with RDF data and that form the core of the Semantic Web stack. The envisaged goal of the **Unifying Logic** layer is as an interoperability layer that provides the foundation for combining these lower-level technologies into a whole, with a unifying language to engage queries and rules over knowledge represented in RDF and associated ontologies/schemata. Various works in this area have looked at combining rules with querying [47, 48], combining ontological interpretations with querying [35, 20], and combining rules and ontologies [37, 32, 36].

Proof: Given that the Semantic Web would enable software agents to perform various automated tasks over decentralized sources of structured information, possibly combining data from multiple external sources and applying various reasoning and querying primitives to achieve some goal, it is important that the software agent provide some form of *proof* that can be used by the client to validate the procedure or information used to, e.g., complete the task or derive an answer.

Trust: Related to the underlying **Proof** layer, the **Trust** layer would be required by clients on the Semantic Web to determine which sources of information should be trusted in a proof, or by clients and servers as an access control mechanism to determine which service providers or other agents on the Web are allowed access to which data, and so forth. To achieve this, the **Trust** layer would not require an a priori whitelist or blacklist of agents, but should rather be able to determine trust for agents it has not seen before based on attributes of that agent (e.g., based on a social network, being a governmental body, etc.).

Cryptography: This layer lies to the side of the Semantic Web stack, indicating that although important, cryptography is somewhat tangential to the core Semantic Web technologies. Obviously, the Semantic Web would require cryptographic techniques for verifying identity and for allowing access control mechanisms, and so forth. However, many existing cryptography technologies could be borrowed directly from the Web, including digital signatures, public-key encryption/decryption algorithms such as RSA, secure protocols such as HTTP Secure (HTTPS) that use TSL/SSL, and so forth.

The original Semantic Web vision [9] is indeed an ambitious one. Throughout the years, various aspects of the stack have been tackled by a variety of research groups, developers and standardization bodies. However, much of the original vision remains unrealized. On a high-level, we see that lower parts of the Semantic Web stack borrow directly from existing Web technologies, middle parts of the stack have been realized through various standardization efforts, and higher parts of the stack remain largely unrealized. In general, however, the stack is best viewed in a descriptive manner, not a prescriptive manner: it is an illustration, not a specification.

Many developments have been in made in the past few years on the middle layers of the stack in terms of the RDF data-model and related standards built on top for querying RDF, representing schemata and ontologies in RDF, and expressing rules that can be executed over RDF data. This book focuses largely on these middle layers, and the remainder of this chapter outlines the core Semantic Web standards that have been proposed in these areas, starting with the RDF standard.

1.3 Resource Description Framework (RDF)

The RDF standard [42] provides the basis for a core agreed-upon data-model on the Semantic Web. Having an agreed-upon data-model is crucial for the interoperability of data produced by different independent publishers across the Web, allowing for content represented in RDF to be generically processed and indexed by off-the-shelf tools no matter what its topic or origin.

Herein, we give a brief walkthrough of the design principles and the features of RDF. We do not cover all features, but rather focus on core concepts that are important for further reading of this book. Throughout, we will use Turtle's syntactic conventions for representing RDF terms and RDF data.

These conventions will be introduced in an incremental fashion, but if unfamiliar with the syntax, the reader may find it worthwhile to look through the examples in the W3C Working Draft for Turtle [5].

1.3.1 RDF Terms

The elemental constituents of the RDF data-model are *RDF terms* that can be used in reference to *resources*: anything with identity. The set of RDF terms is broken down into three disjoint sub-sets: *URIs* (or *IRIs*), *literals* and *blank nodes*.[2]

URIs serve as global (Web-scope) identifiers that can be used to identify any resource. For example, `http://dbpedia.org/resource/Lemon` is used to identify the lemon fruit (and plant) in DBpedia [12] (an online RDF database extracted from Wikipedia content). In Turtle, URIs are delimited with angle-brackets: `<http://dbpedia.org/resource/Lemon>`. To avoid writing long and repetitive full URI strings, Turtle allows for the use of CURIE-style shortcuts [11] where a re-usable prefix can be defined: `@prefix dbr: <http://dbpedia.org/resource/>`. Thereafter, URIs can be abbreviated using `prefix:localname` shortcuts—e.g., `dbr:Lemon`—where the local-name is resolved against the in-scope prefix definition to generate the full URI form.

Literals are a set of lexical values denoted with inverted commas in Turtle. Literals can be either:

Plain Literals which form a set of plain strings, such as `"Hello World"`, potentially with an associated language tag, such as such as `"Hello World"@en`.

Typed Literals which comprise of a lexical string and a datatype, such as `"2"^^xsd:int`. Datatypes are identified by URIs (such as `xsd:int`), where RDF borrows many of the datatypes defined for XML Schema that cover numerics, booleans, dates, times, and so forth. These datatypes define which lexical forms are valid for that datatype (e.g., to state that `"p"^^xsd:int` is invalid), and ultimately provide a mapping from valid lexical strings to a value space (e.g., from `"2"^^xsd:int` to the value of the number two). Turtle provides shortcuts for common datatypes, where the use of numbers and boolean values without quotes—e.g., `2`, `2.4`, `false`—indicate a corresponding datatype literal. Plain literals without language tags map to the same value as lexically identical `xsd:string` values.

[2] Although support for IRIs is featured in more recent RDF-based standards, to avoid confusion, we henceforth stick with the notion of URIs in our discussion and definitions. The distinction between URIs and IRIs is not important to the discourse presented.

Blank Nodes are defined as existential variables used to denote the existence of some resource without having to explicitly reference it using a URI or literal. In practice, blank nodes serve as locally-scoped identifiers for resources that are not otherwise named. Blank nodes cannot be referenced outside of their originating scope (e.g., an RDF document). The labels for blank nodes are thus only significant within a local scope. Intuitively, much like variables in queries, the blank nodes of an RDF document can be relabeled (bijectively) without affecting the interpretation of the document. In Turtle, blank nodes can be referenced explicitly with an underscore prefix `_:bnode1`, or can be referenced implicitly (without using a label) in a variety of other manners.

We can now provide formal notation for referring to the different sets of RDF terms:

Definition 1. *The set of RDF terms is the union of three pair-wise disjoint sets: the set of all URIs (*$\mathbf{U}$*), the set of all literals (*$\mathbf{L}$*) and the set of all blank nodes (*$\mathbf{B}$*). The set of all literals can be further decomposed into the union of two disjoint sets: the set of plain literals (*$\mathbf{L}_p$*) and the set of typed literals (*$\mathbf{L}_t$*). Note that* $\mathbf{U} \cap \mathbf{B} \cap \mathbf{L}_p \cap \mathbf{L}_t = \emptyset$.

Importantly, RDF does not take the Unique Name Assumption (UNA): two terms can (and often do) refer to the same referent. Since RDF is intended to be used as a common data model for the Web, it is likely that two different publishers may use different terms to refer to the same thing. For example, the URI `http://rdf.freebase.com/ns/m.09k_b` is used by Freebase—another online publisher of RDF—to identify the lemon fruit/tree. Thus, by not taking the UNA, RDF and its related standards allow for data from the DBpedia and Freebase exporters to be merged without *requiring* that terms map bijectively to referents. In fact, although RDF terms are composed of pair-wise disjoint sets, different types of RDF terms can also refer to the same thing. For example, the URI `http://km.aifb.kit.edu/projects/numbers/web/n2` is used by the Linked Open Numbers project [59] to assign a URI to the number 2, thus referring to the same value as the term `"2"^^xsd:integer` although the actual terms themselves are in disjoint sets.

1.3.2 RDF Triples and Graphs

Having covered the RDF terms used to refer to things, we now cover *RDF triples* which are used to make statements about those things. The notion of RDF triples constitutes the foundation of the Semantic Web's core data model. As its name suggests, an RDF triple is simply a 3-tuple of RDF terms. The first element of the tuple is called the *subject*, the second element the *predicate*, and the third element the *object*. An RDF triple can then be seen as representing an atomic "fact" or a "claim". Importantly, RDF triples have fixed arity (length of three) with fixed slots (subject, predicate, object), constituting a generic common framework that enables interoperability.

We can formally define the notion of an RDF triple as follows:

Definition 2. *An RDF triple t is defined as a triple $t = (s, p, o)$ where $s \in \mathbf{U} \cup \mathbf{B}$ is called the* subject*, $p \in \mathbf{U}$ is called the* predicate *and $o \in \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$ is called the* object*.*

Informatively, the typical role of the three RDF triple positions can be intuited as follows:

Subject: Filled by an RDF term (either a URI or a blank node) that refers to the *primary resource* being described by the triple.

Predicate: Filled by an RDF term (must be a URI) that identifies the *relation* between the subject and the object.

Object: Filled by an RDF term (can be a URI, blank node or literal) that fills the *value* of the relation.

How RDF triples can then be used to describe resources is best illustrated with an example:

Example 1. *The following example presents some of the RDF data talking about lemons and about citrus from the DBpedia exporter ('*`#`*' denotes a comment line in Turtle):*

```
# PREFIX DECLARATIONS
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# RDF TRIPLES
dbr:Lemon rdfs:label "Lemon"@en .
dbr:Lemon dbp:calciumMg 26 .
dbr:Lemon dbo:family dbr:Rutaceae .
dbr:Lemon dbo:genus dbr:Citrus .
dbr:Citrus rdfs:label "Citrus"@en .
dbr:Citrus dbo:family dbr:Rutaceae .
dbr:Citrus dbo:family dbr:Aurantioideae .
```

Here we see four prefix declarations and then seven RDF triples (delimited by periods in Turtle). Each triple is comprised of three RDF terms. The subject position contains URIs (or blank nodes) that identify what can be viewed as the primary resource being described (in this case, `dbr:Lemon` *and* `dbr:Citrus`*). The predicate position contains a URI that identifies the relation (aka. attribute) being described for that resource (e.g.,* `rdfs:label`*,* `dbo:genus`*). The object position contains URIs and literals (and potentially blank nodes) that refer to the value for that relation (e.g.,* `26`*,* `dbr:Citrus`*).*

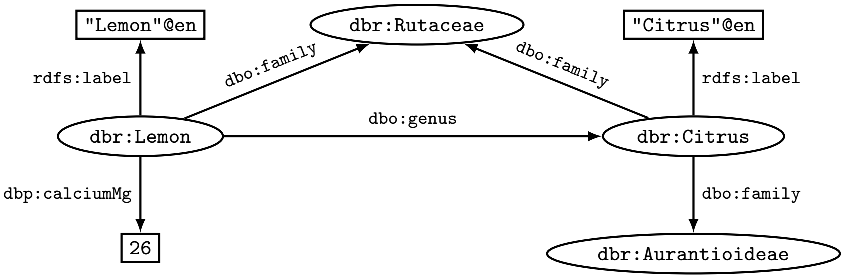
Turtle permits some abbreviations when writing triples that contain repetitions. Omitting the prefix declarations for brevity, the following triples represent the same content as above:

```
...
# RDF TRIPLES
dbr:Lemon rdfs:label "Lemon"@en ;
   dbp:calciumMg 26 ;
   dbo:family dbr:Rutaceae ;
   dbo:genus dbr:Citrus .
dbr:Citrus rdfs:label "Citrus"@en ;
   dbo:family dbr:Rutaceae , dbr:Aurantioideae .
```

Here, '`;`*' indicates that the subsequent triple has the same subject as the previous triple, allowing to omit that term. Also, '*`,`*' indicates that the subsequent triple contains the same subject and predicate as the previous triple, allowing to omit those terms.*

It is common practice to conceptualize RDF datasets as directed labeled graphs, where subjects and objects are drawn as labeled vertices and predicates are drawn as directed, labeled edges. By convention, literal vertices are drawn as rectangles, and URI vertices and blank nodes are drawn as ellipses (labels for blank node vertices are often omitted).

Example 2. *The following diagram renders the above RDF dataset as a directed labeled graph,*



It is worth nothing that the RDF data-model is not directly isomorphic with the notion of directed labeled graphs. In particular, edge labels can themselves be vertices, which cannot be represented in such a diagram. This occurs when a predicate term also appears in a subject or object position; for example, if one were to add the following triple to the data in question:

```
...
dbo:genus rdfs:label "Genus"@en .
```

one would either need to duplicate the `dbo:genus` *term for both a vertex and an edge, or to allow extending edges between edges. Both remedies would break the standard formal conventions for a directed labeled graph.*

Although some authors have suggested alternative representations such as bipartite graphs for RDF [28], directed labeled graphs remain an intuitive and popular conceptualization of RDF data.[3] As such, RDF is often referred to as being *graph-structured data* where each (s, p, o) triple can be seen as an edge $s \xrightarrow{p} o$. In fact, a set of RDF triples is formally referred to as an *RDF graph*.

Definition 3. *A finite set of RDF triples* $G \subset (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ *is called an* RDF graph.

Since RDF graphs are defined in terms of sets, it follows that the ordering of RDF triples in an RDF graph is entirely arbitrary and that RDF graphs do not allow for duplicate triples. The graph-structured nature of the RDF data-model lends itself to flexible integration of datasets. Additional edges can be added extensibly to the graph at any time. Edges in the graph use globally-scoped URI identifiers. When vertices are identified with URIs, they too can be referenced externally and connected to other vertices.

There is one slight complication in the notion of RDF graphs, caused by blank nodes. Blank nodes are intended to be locally-scoped terms that are interpreted as existential variables—as denoting the existence of something without naming it. The labels of blank nodes are not of significance outside of the local scope. This gives rise to a notion of *isomorphism* between RDF graphs that are the same up to (bijective) blank-node relabeling: isomorphic RDF graphs can be considered as containing the same "content".[4] Furthermore, when merging two (or more) RDF graphs, it is important to ensure that there are no conflicts in blank-node labels. If two RDF graphs share a blank node with the same label, that blank node is not considered the same across the two graphs. Hence, the notion of an *RDF merge* is introduced to avoid blank-node label conflicts.

Definition 4. *Given two RDF graphs,* G_1 *and* G_2*, an* RDF merge *of these two graphs, denoted* $G_1 \uplus G_2$*, is defined as the set union* $G'_1 \cup G'_2$ *where* G'_1 *and* G'_2 *are isomorphic copies of* G_1 *and* G_2 *respectively such that the copies do not share any blank nodes with common labels.*

The existential nature of blank-nodes also gives rise to the notion of *(non-)lean* RDF graphs, whereby non-lean graphs contain redundant blank nodes [29]. We do not go into detail on this particular subject as it relates to an often overlooked and rarely relevant aspect of RDF [41], but rather make the reader aware of the issue with an illustrative example.

Example 3. *The following dataset features two blank nodes (omitting prefix declarations for brevity where* `ex:` *refers to an arbitrary example namespace):*

```
ex:LemonPieRecipe ex:ingredient dbr:Lemon .
```

[3] Where a predicate also appears in the subject or object position, most commonly, the term is duplicated as a vertex and an edge.

[4] An analogy would be to consider two queries that are the same up to variable relabeling.

```
dbr:Lemon rdfs:label "Lemon"@en .
ex:LemonPieRecipe ex:ingredient _:bn1 .
_:bn1 rdfs:label "Lemon"@en .
ex:LemonPieRecipe ex:ingredient _:bn2 .
```

Analogously, since the labels of blank nodes are not important other than in a local scope, Turtle permits using '`[]`*' as a shortcut to represent blank nodes:*

```
ex:LemonPieRecipe ex:ingredient dbr:Lemon .
dbr:Lemon rdfs:label "Lemon"@en .
ex:LemonPieRecipe ex:ingredient [ rdfs:label "Lemon"@en ] .
ex:LemonPieRecipe ex:ingredient [] .
```

Both of these data snippets represent isomorphic RDF graphs, either of which can be loosely read as stating the following:

1. *The first two triples state that the recipe* `ex:LemonPieRecipe` *has the ingredient* `dbr:Lemon`*, which has the label* `"Lemon"@en`*.*
2. *The third and fourth triples state that the same recipe has* some *ingredient, which has the label* `"Lemon"@en`*.*
3. *The fifth triple states that the same recipe has* some *ingredient.*

Here, point 3 is made redundant by knowledge of points 1 & 2, and point 2 is made redundant by knowledge of point 1. Thus the RDF graph represented by the above triples can be considered non-lean*. The lean version of this graph—containing no redundancy due to existential blank nodes—would then be:*

```
ex:LemonPieRecipe ex:ingredient dbr:Lemon .
dbr:Lemon rdfs:label "Lemon"@en .
```

Both the lean and the non-lean versions can be considered as containing the same core information. A common misconception would be to view the original RDF graph as indicating the presence of three ingredients. However, it is important to remember: (1) RDF does not take the UNA, (2) blank nodes do not identify particular things, only the existence of things.

1.3.3 RDF Vocabulary

The notions of RDF triples and RDF graphs thus form the core of the RDF data model. In addition, the RDF standard provides a set of "built-in" vocabulary terms under a core RDF *namespace* (a common URI prefix scheme) that standardize popular RDF patterns. We do not cover all of the built-in RDF vocabulary terms, instead covering the most prevalent features.

The most popular term in the RDF vocabulary is `rdf:type`, which is used to assign resources sharing certain commonalities into *classes*.

Example 4. *The following data assigns six instances to five different classes:*

```
# PREFIX DECLARATIONS
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
...

# RDF TRIPLES
ex:LemonPieRecipe rdf:type ex:Recipe .
ex:RisottoRecipe rdf:type ex:Recipe .
dbr:Lemon rdf:type dbo:Plant , dbo:Eukaryote .
dbr:Citrus rdf:type dbo:Plant , dbo:Species .
dbo:genus a rdf:Property .
dbo:order a rdf:Property .
```

Resources can be instances of multiple classes and classes can have multiple instances. As illustrated by the last two triples, Turtle syntax allows for using "`a`*" as a simple shortcut for the URI* `rdf:type`*. Furthermore, the last two triples contain the* `rdf:Property` *class: a built-in RDF class used to denote the set of all* properties *(URI terms used as relations that appear in the predicate position of triples).*

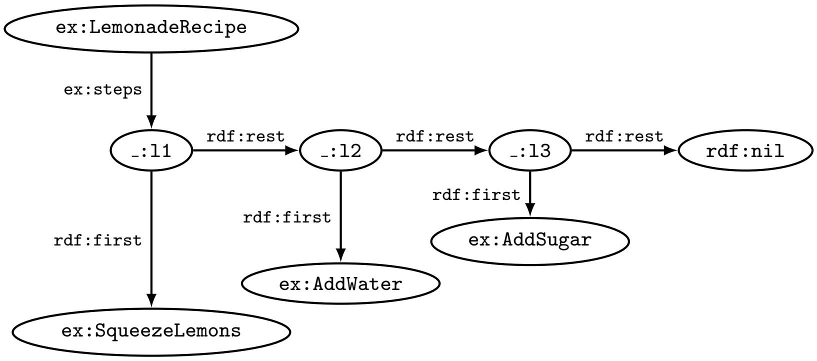
The `rdf:type` term is by far the most frequently used of the built-in RDF vocabulary. As we will see later in the next section, the semantics of classes introduced by use of the `rdf:type` relation can be defined using the RDFS and OWL standards.

Another quite widely used feature of RDF is its vocabulary for describing *RDF collections* (aka. *RDF lists*). Since the set-based RDF data-model has no inherent ordering, RDF collections can be used to define an ordered (and closed) list using a linked-list pattern. RDF standardizes an agreed-upon vocabulary and structure for defining such lists.

Example 5. *The following is an RDF graph containing an ordered collection of steps for a recipe.*

```
ex:LemonadeRecipe ex:steps _:l1 .
_:l1 rdf:first ex:SqueezeLemons .
_:l1 rdf:rest _:l2 .
_:l2 rdf:first ex:AddWater .
_:l2 rdf:rest _:l3 .
_:l3 rdf:first ex:AddSugar .
_:l3 rdf:rest rdf:nil .
```

These triples state that the resource `ex:LemonadeRecipe` *has a set of steps, which is represented as an RDF collection containing three elements. The RDF collection itself is essentially a linked list. The following diagram illustrates this linked list structure.*



Each of the blank nodes `_:l1`*,* `_:l2` *and* `_:l3` *represent a (sub-)list with two outgoing relations:* `rdf:first` *indicates the single element attached to that (sub-)list and* `rdf:rest` *connects to the subsequent (sub-)list. The list is terminated with the built-in* `rdf:nil` *term, which indicates an empty list. Though not enforced by the RDF standard, (sub-)lists are typically represented by blank nodes and—with the exception of* `rdf:nil`*—have precisely one value for* `rdf:first` *that can be any RDF term, and one value for* `rdf:rest`*. This structure provides two main benefits: (1) the list is ordered, (2) the list is closed.*

Turtle provides a convenient shortcut syntax using '`()`*' to indicate a list:*

```
ex:LemonadeRecipe ex:steps ( ex:SqueezeLemons ex:AddWater ex:AddSugar ) .
```

This Turtle snippet serializes an RDF graph isomorphic with the full form represented above.

The RDF vocabulary also provides support for a number of other features that are not often used and that we do not discuss here in detail:

RDF Containers offer an alternative to collections for specifying either ordered or unordered lists in RDF. However, unlike collections, containers cannot be closed.

RDF ***n*****-ary Predicates** provide a standard mechanism for specifying complex relationships in RDF. For example, one may wish to not only state that a recipe has the ingredient lemon, but also to state the quantity of lemon it contains. This can be achieved using *n*-ary predicates whereby a new RDF resource is created to represent and describe the relation itself.

RDF Reification provides a method to talk about individual RDF triples themselves within RDF. The method works by creating a new resource that refers to an RDF triple, stating what subject, predicate and object it has, and then adding additional information about that RDF triple.

We do not cover these standard RDF features in detail since they are rarely used and, in fact, there have been calls to deprecate such features in future versions of RDF [61]. For more about the above features, we instead refer the interested reader to the RDF primer [42].

1.3.4 RDF Syntaxes

There are a number of syntaxes available for writing RDF data down—for serializing RDF data. Thus far, we have been using the Turtle syntax for examples: Turtle is perhaps the most human-readable syntax available for encoding RDF data. However, there are a variety of options available.

RDF/XML [4] is one of the oldest and most established syntactic representations for RDF, having been standardized early on [39] (and later revised [4]). As its name suggests, RDF/XML involves encoding RDF in the XML format. The core rationale behind RDF/XML was to leverage already mature XML tools for creating and parsing RDF serializations.[5] RDF/XML remains one of the most widely RDF syntaxes in use today: for example, the SPARQL and OWL standards only require RDF/XML input/output support for full compliance.

Turtle [5] is a custom syntax for RDF based on the related Notation3 (N3) format [8] from which it drops features that go beyond RDF. Turtle aims at a concise and intuitive syntax for representing RDF, with shortcuts for commonly used RDF features. The use of Turtle is not as popular as RDF/XML, primarily because at the time of writing, Turtle has yet to be standardized (though it should be soon [5]). However, Turtle also forms the basis of the SPARQL query syntax. Due to its concise and readable nature, and its relation to SPARQL, this book will primarily use Turtle syntax for examples.

N-Triples [23] is a simple syntax for RDF and a proper subset of Turtle: all N-Triples files are valid Turtle. N-Triples disallows all forms of Turtle shortcuts, requiring that all RDF terms are written in their full form; for example, prefixes are not allowed where URIs must instead be written in full using `<>` delimiters. Triples must be written in full and delimited by newlines. Thus, in N-Triples, each individual line contains all the necessary information to parse the triple on that line independent of the rest of the document. This makes N-Triples popular for streaming applications and for fault-tolerant line-at-a-time processing.

RDFa [2, 1] provides syntax for embedding RDF data into (X)HTML documents. The core rationale behind RDFa is to allow for embedding RDF

[5]... at least on a syntax level. Since RDF/XML cannot offer a canonical XML representation of RDF, technologies such as XQuery and XPath that are designed to navigate and process XML trees had little application for RDF other than on a syntax level.

data intended for machine consumption into HTML documents intended for human consumption, avoiding the need to have separate documents for each. Having one document for both machines and humans simplifies hosting RDF data, particularly aspects like content negotiation, etc. The original RDFa recommendation [2] has been superseded by the RDFa 1.1 Core standard [1], which makes a variety of changes primarily aiming to remove the reliance on XHTML and to make the original syntax more usable for Web developers. Relatedly, an RDFa 1.1 Lite version—a lightweight, easy-to-learn and frequently sufficient subset of RDFa 1.1 Core—has also been standardized, aimed at Web developers.

JSON LD [56] is a JSON-based syntax for representing data in a format that equates largely to RDF. Since JSON is widely used as a serialization format by many Web applications, JSON LD would then allow Web developers to parse RDF using the legacy JSON parsing mechanisms available in the scripting languages of their choice, and, for example, allow for handling RDF graphs as Javascript objects. Being a relatively recent proposal, JSON LD is currently a W3C Working Draft [56].

No matter what syntax is chosen, the data are represented in the same RDF data model: it is possible to convert directly from RDF in one syntax to another, and although the result may not be canonical, both input and output will represent the same RDF data.[6]

1.4 RDF Semantics, Schemata and Ontologies

Thus far, we have covered structural aspects of RDF. As the outset of this chapter, we also highlighted the importance of semantics for providing machines with an ‘interpretation" of the data they process, allowing them to perform automated tasks based on the content in front of them. This section outlines the semantics of RDF itself, as well as two other related standards that extend RDF with richer semantics: RDFS and OWL.

1.4.1 RDF Semantics

The semantics of RDF has been standardized in the form of a *model theory* [29], which, in this case, uses mathematical concepts to provide a formal foundation for the machine-processable meaning of RDF data. The details of the model theory underpinning RDF Semantics are not important for our

[6] There are very minor exceptions to this rule: for example, RDF/XML cannot encode certain RDF graphs that contain predicates that cannot be referenced using QName conventions.

purposes and are thus considered out of scope—we instead refer the reader to the RDF Semantics standard [29]. To summarize, model theory introduces the notion of *worlds* that serve as *interpretations* of RDF data, whereby RDF triples are seen as making claims about the nature or configuration of that world. The more that is claimed about the world through RDF, the more specific the world becomes in order to make those RDF claims true, and thus the narrower the field of possible interpretations becomes. The mechanics of the interpretation of RDF graphs provides the mathematical basis for stating what the URIs used in the RDF data identify in the world, what things in the world are related through which properties, what value individual (datatype) literals map to, what existential blank nodes represent, and so forth.

Importantly, the semantics of RDF do not make machines any more aware of the real world than they are now. The RDF Semantics does *not* try to capture the full intended meaning of the RDF data, which may require background or contextual knowledge, common sense, ability to read natural language annotations, or to know what a "`Lemon`" is and perhaps to have tasted one, etc. Instead, the aim is to formalize a well-defined subset of the semantics of RDF data such that it can be leveraged by machines to automate various tasks: to formalize claims about the world in a well-understood manner that allows for evaluating the consistency of claims, the necessary *entailments* of those claims, and ultimately to evaluate the truth of those claims according to the theory.

The most important notion arising from the RDF Semantics standard is the aforementioned notion of entailment, which when given an RDF graph containing various claims that are held as true, formalizes what other claims are, as a consequence, also held true (or indeed, false). This provides a foundation for machine-readability, which ultimately avoids the need to redundantly write down all things that are true, instead letting machines "join the dots".

Example 6. *Take the simple RDF graph containing two claims:*

```
dbr:Lemon dbo:family dbr:Rutaceae .
dbr:Citrus dbo:family dbr:Rutaceae .
```

This RDF graph trivially entails all of its sub-graphs (i.e., subsets), including itself. Due to the existential nature of blank nodes, it also entails the following RDF graph:

```
[] dbo:family dbr:Rutaceae .
dbr:Lemon dbo:family [] .
dbr:Citrus dbo:family [] .
[] dbo:family [] .
```

This graph states that there is something which has the family `dbr:Rutaceae`*, that* `dbr:Lemon` *and* `dbr:Citrus` *have some family, and that there exists a family relation between some two things. All of these claims are necessarily* *held true if the original two triples are held true. Furthermore, the last triple of the second graph is entailed by any of the three triples above it, making the graph non-lean.*

The form of entailment illustrated in the previous example—involving either sub-graphs or existential blank-nodes—is called *simple entailment*. Built on top of simple entailment is *RDF entailment*. RDF entailment further includes a couple of straightforward entailments for RDF graphs.

Example 7. *Take the following single-triple RDF graph:*

```
dbr:Lemon dbo:family dbr:Rutaceae .
```

This entails:

```
dbo:family a rdf:Property .
```

In RDF, any term appearing in the predicate position of a triple is interpreted as refering to a property, an instance of the class `rdf:Property`*.*

This example provides semantics for the `rdf:Property` class in the RDF vocabulary, and is covered under RDF entailment (but not simple entailment). Entailments such as these can be automatically realized through use of *inference rules*, which match premises in an RDF graph and use these premises to derive novel conclusions. An inference rule to support the previous form of entailment might look like (in Turtle-style syntax):

```
?s ?p ?o . ⇒ ?p a rdf:Property .
```

Variables are prefixed with '`?`' and can be matched by any RDF term.[7] The left of the rule is called the *body* (aka. *antecedent*) of the rule and matches premises in the graph. The right side of the rule is called the *head* (aka. *consequent*) of the rule and provides a template for inferences based on the matched premises.[8] In the inference rules for RDF and related standards, variables appearing in the head of the rule must appear in the body. Every claim or set of claims matched by the body entail the corresponding consequences produced by the head (under the same variable substitution).

If entailment is the formal theory on what conclusions can follow from what

[7] The entailment rules defined for the RDF Semantics documentation restrict what types of terms some variables can be matched by to ensure the inference of only valid RDF triples, but this is not an important detail and is overlooked by many implementations that allow *generalized triples*—which relax the restrictions on the types of terms allowed in triple positions—in intermediate inferences.

[8] In the RDF world, rules are often definite Horn clauses and are typically a syntactic subset of Datalog (using atoms that apply for RDF triples). Inference rules are sometimes written in the other direction, with the head first, a left-arrow, and then the body.

premises, then inference can be thought of as the implementation of entailment, and as a form of (deductive) reasoning. Henceforth, we primarily stick with the notion of inference rules to explain the semantics of RDF and its related standards.[9] Since the RDF vocabulary is just a foundation and not very expressive—it does not contain many well-defined terms—RDF entailment in itself is still pretty boring and not very useful for practical applications.[10] However, two further standards have been created to add additional well-defined vocabulary for making claims using RDF: the RDF Schema (RDFS) and Web Ontology Language (OWL) standards. In the following sub-sections, we elaborate further upon these two standards.

1.4.2 RDF Schema (RDFS)

In April 1998, the first draft of the RDF Schema (RDFS) specification was published as a W3C Working Note [14]. The core idea was to extend upon the RDF vocabulary and allow for attaching semantics to user-defined classes and properties. The original proposal was to be heavily modified in later versions; for example, features relating to database-like constraints were dropped in favour of a specification more explicitly in tune with the Open World Assumption. The modern RDFS specification thus became a W3C Recommendation in early 2004 [13] along with the description of the RDF Semantics [29] discussed thus far. RDFS extends RDF with four key terms [45] that allow for specifying well-defined relationships between classes and properties:

`rdfs:subClassOf` (**sC**) allows for stating that the extension of one class c_1 (its set of members) is necessarily contained within the extension of another class c_2.

`rdfs:subPropertyOf` (**sP**) allows for stating that all things related by a given property p_1 are also necessarily related by another property p_2.

`rdfs:domain` (**dom**) allows for stating that the subject of a relation with a given property p is a member of a given class c.

`rdfs:range` (**rng**) analogously allows for stating that the object of a relation with a given property p is a member of a given class c.

The RDFS vocabulary does contain other terms not highlighted here, but these remaining terms are largely syntactic elements that do not yield particularly useful entailments. One important term that should be mentioned however is `rdfs:Resource`, which refers to the class of all resources in RDFS.

[9] Inference rules are sufficient to support the complete semantics of RDF and RDFS, with minor exceptions. However, the (two) semantics of OWL are more complex and cannot be *completely* supported by a finite set of inference rules alone.

[10] Aside from instantiating the `rdf:Property` class, RDF entailment also provides for interpreting an RDF datatype that represents valid XML strings, which does little to add to our excitement for RDF entailment by itself.

TABLE 1.1: A *selection* of RDF(S) rules (see [29, §7] for all).

Rule ID	Body	Head
rdf1	?s ?p ?o .	⇒ ?p a rdf:Property .
rdfs2	?p dom ?c . ?x ?p ?y .	⇒ ?x rdf:type ?c .
rdfs3	?p rng ?c . ?x ?p ?y .	⇒ ?y rdf:type ?c .
rdfs5	?p$_1$ sP ?p$_2$. ?p$_2$ sP ?p$_3$.	⇒ ?p$_1$ sP ?p$_3$.
rdfs7	?p$_1$ sP ?p$_2$. ?x ?p$_1$?y .	⇒ ?x ?p$_2$?y .
rdfs9	?c$_1$ sC ?c$_2$. ?x rdf:type ?c$_1$.	⇒ ?x rdf:type ?c$_2$.
rdfs11	?c$_1$ sC ?c$_2$. ?c$_2$ sC ?c$_3$.	⇒ ?c$_1$ sC ?c$_3$.

This class can be thought of as the "universal class" containing everything, including literals, classes, properties—and even itself. Furthermore, the class `rdfs:Class` is defined, which refers to the class of all classes (including itself).

This RDFS vocabulary is then formally defined by the RDF Semantics documents in terms of the RDF model theory, which in turn gives rise to the notion of RDFS entailment, which is layered on top of RDF entailment, and which allows for defining a set of RDFS inference rules [29]. A selection of the most important RDF(S) inference rules are listed in Table 1.1 for reference, using the shortcuts for RDFS terms outlined previously; the full list is available in the RDF Semantics document [29, §7] and supports the complete semantics of RDFS.[11]

Example 8. *The following RDF graph:*

```
dbr:Citrus a dbo:FloweringPlant .
dbr:Lemon dbo:genus dbr:Citrus .
dbo:FloweringPlant rdfs:subClassOf dbo:Plant .
dbo:Plant rdfs:subClassOf dbo:Eukaryote .
dbo:genus rdfs:domain dbo:Species .
dbo:Species rdfs:subClassOf owl:Thing .
```

RDFS-entails (amongst other triples):

```
dbo:FloweringPlant rdfs:subClassOf dbo:Eukaryote . #rdfs11
dbr:Citrus a dbo:Plant , dbo:Eukaryote . #rdfs9
dbr:Lemon a dbo:Species , owl:Thing . #rdfs2
```

The comment for each entailed triple denotes a rule from Table 1.1 by which it can be inferred. First note that domains and ranges do not act as constraints: although the domain of `dbo:genus` *is defined as* `dbo:Species`*, it is not considered a problem that* `dbr:Lemon` *is not stated to be a member of that class, where instead, it is simply inferred to be a member of that class. Second note that, as per the final inferred triple, inferences can be applied recursively to find further valid entailments.*

[11] There are some corner-cases and bugs in the RDF Semantics that lead the inference rules to be incomplete [45, 58], but these are not important for our purposes.

RDFS entailment is layered on top of RDF entailment, which is in turn layered on top of simple entailment. Another form of entailment specified by the RDF Semantics document is datatype entailment or *D-entailment*. The core purpose of D-entailment is to formalize, for a set of pre-defined datatypes, a map from lexical strings to the values that they denote: for example, to map `"2.0"^^xsd:decimal` to the value of the number two. D-entailment is most commonly used in conjunction with XSD datatypes [29, §5].

1.4.3 Web Ontology Language (OWL)

Evolving from earlier proposals for Web ontology languages—such as that of SHOE [40], DAML [31], OIL [17] and the subsequent hybrid DAML+OIL [33]—in 2001 the W3C began working on a new ontological language that would extend upon RDFS with more expressive semantics, enabling richer entailment regimes. In 2004, the resulting Web Ontology Language (OWL) was recognized as a W3C Recommendation [43]. This was subsequently extended in 2008 by the OWL 2 W3C Recommendation [22].

Relative to RDFS, OWL is a much more complicated standard, with a deep and colorful background rooted in various competing academic proposals, resulting in a standard that spans a plethora of W3C documents. It is not necessary for us to faithfully replicate the complexity of the OWL standard and its background here. Instead, we focus on a high-level overview, and focus on details only for pertinent aspects.

Like RDFS, OWL can also be serialized as RDF triples. In fact, OWL re-uses the core RDFS vocabulary as described in the previous section (with analogous semantics), but adds a wealth of new vocabulary rooted in a well-defined semantics. We now summarize a small subset of the novel features provided by OWL (2).[12]

`owl:equivalentClass` **(eC)** allows for stating that two classes have the same extension (e.g., the classes `Human` and `Person` are said to have the same members).

`owl:disjointWith` **(dC)** allows for stating that the extensions of two classes have an empty intersection (e.g., the classes `Human` and `Flower` cannot share members).

`owl:equivalentProperty` **(eP)** allows for stating that two properties relate precisely the same things (e.g., the properties `parentOf` and `hasChild` are said to relate the same resources).

`owl:disjointPropertyWith` **(dP)** allows for stating that two properties can never relate the same two things (e.g., the properties `brotherOf` and `sisterOf` cannot relate the same two people in the same direction).

[12] The OWL features introduced correspond loosely to the most prominently used features on the Web of Data [19].

`owl:inverseOf` **(inv)** allows for stating that one property relates the same things as another property, but in the opposite direction (e.g., `parentOf` and `childOf` are said to relate the same resources, but in inverse directions).

`owl:TransitiveProperty` **(TP)** is a class whereby properties that are a member specify a transitive relation (e.g., `ancestorOf` is a transitive relation where the ancestor of one's ancestor is also always one's ancestor, and so forth).

`owl:SymmetricProperty` **(SP)** is a class whereby properties that are a member specify a symmetric (bidirectional) relation (e.g., `siblingOf` is symmetric since one is always a sibling to one's sibling).

`owl:sameAs` **(sA)** allows for stating that two resources refer to the same thing, and that the information for one resource applies equally to the other (useful when two URIs are used in RDF to refer to the same thing).

`owl:differentFrom` **(dF)** allows for stating that two resources necessarily refer to different things (cannot be in an `owl:sameAs` relation).

`owl:FunctionalProperty` **(FP)** allows for stating that a subject resource can only have one value for that property (e.g., `hasBiologicalFather` since there can only be one biological father for a person). If a subject resource has two object values for such a property, those two objects must be coreferent (i.e., be in an `owl:sameAs` relation).

`owl:InverseFunctionalProperty` **(IFP)** allows for stating that the value of a property is unique to the subject of the relation (e.g., `isbn` values uniquely identify books, `biologicalFatherOf` identifies a biological father by his child as the inverse of the previous case, and so forth). If two subjects share the same object for a given inverse-functional property, those two subjects are coreferent.

An example list of rules corresponding to these OWL (2) features are listed in Table 1.2 for reference (these are a subset of OWL 2 RL/RDF rules [24], which will be briefly introduced later). One can also see some overlap with the RDFS rules presented previously in Table 1.1: as previously stated, OWL builds on top of parts of the RDFS vocabulary. Furthermore, the head of some rules consists of the lone symbol `false`, which is used to indicate that data matching the body of the rule in question forms an *inconsistency*: a logical contradiction that according to the OWL semantics, indicates a formal error in the data.

We now give a brief example of one feature of OWL that is frequently used in RDF data published on the Web: `owl:sameAs` [19, 26].

Example 9. *Again, we take some example data about Citrus from the DBpedia exporter:*

TABLE 1.2: A *selection* of OWL 2 RDF/RDF rules [24, §4.3].

Rule ID	Body	Head
eq-sym	?x sA ?y .	⇒ ?y sA ?x .
eq-trans	?x sA ?y . ?y sA ?z .	⇒ ?x sA?z .
eq-rep-s	?s sA ?s$'$. ?s ?p ?o .	⇒ ?s$'$?p ?o .
eq-rep-p	?p sA ?p$'$. ?s ?p ?o .	⇒ ?s ?p$'$?o .
eq-rep-o	?o sA ?o$'$. ?s ?p ?o .	⇒ ?s ?p ?o$'$.
eq-diff1	?x sA ?y . ?x dF ?y .	⇒ FALSE
prp-dom	?p dom ?c . ?x ?p ?y .	⇒ ?x a ?c .
prp-rng	?p rng ?c . ?x ?p ?y .	⇒ ?y a ?c .
prp-fp	?p a FP. ?x ?p ?y$_1$, ?y$_2$.	⇒ ?y$_1$ sA ?y$_2$.
prp-fp	?p a IFP. ?x$_1$?p ?y . ?x$_2$?p ?y .	⇒ ?x$_1$ sA ?x$_2$.
prp-symp	?p a SP. ?x ?p ?y .	⇒ ?y ?p ?x .
prp-trp	?p a TP. ?x ?p ?y . ?y ?p ?z .	⇒ ?x ?p ?z .
prp-spo1	?p$_1$ sP ?p$_2$. ?x ?p$_1$?y .	⇒ ?x ?p$_2$?y .
prp-eqp1	?p$_1$ eP ?p$_2$. ?x ?p$_1$?y .	⇒ ?x ?p$_2$?y .
prp-eqp2	?p$_1$ eP ?p$_2$. ?x ?p$_2$?y .	⇒ ?x ?p$_1$?y .
prp-pdw	?p$_1$ dP ?p$_2$. ?x ?p$_1$?y ; ?p$_2$?y .	⇒ FALSE
prp-inv1	?p$_1$ inv ?p$_2$. ?x ?p$_1$?y .	⇒ ?y ?p$_2$?x .
prp-inv2	?p$_1$ inv ?p$_2$. ?x ?p$_2$?y .	⇒ ?y ?p$_1$?x .
cax-sco	?c$_1$ sC ?c$_2$. ?x a ?c$_1$.	⇒ ?x a ?c$_2$.
cax-eqc1	?c$_1$ eC ?c$_2$. ?x a ?c$_1$.	⇒ ?x a ?c$_2$.
cax-eqc2	?c$_1$ eC ?c$_2$. ?x a ?c$_2$.	⇒ ?x a ?c$_1$.
cax-dw	?c$_1$ dC ?c$_2$. ?x a ?c$_1$, ?c$_2$.	⇒ FALSE
scm-sco	?c$_1$ sC ?c$_2$. ?c$_2$ sC ?c$_3$.	⇒ ?c$_1$ sC ?c$_3$.
scm-eqc1	?c$_1$ eC ?c$_2$	⇒ ?c$_1$ sC?c$_2$. ?c$_2$ sC ?c$_1$.
scm-eqc2	?c$_1$ sC ?c$_2$. ?c$_2$ sC ?c$_1$.	⇒ ?c$_1$ eC ?c$_2$.
scm-spo	?p$_1$ sP ?p$_2$. ?p$_2$ sP ?p$_3$.	⇒ ?p$_1$ sP ?p$_3$.
scm-eqp1	?p$_1$ eP ?p$_2$	⇒ ?p$_1$ sP?p$_2$. ?p$_2$ sP ?p$_1$.
scm-eqp2	?p$_1$ sP ?p$_2$. ?p$_2$ sP ?p$_1$.	⇒ ?p$_1$ eC ?p$_2$.

```
# PREFIXES
@prefix fb: <http://rdf.freebase.com/ns/> .
...

dbr:Lemon dbo:genus dbr:Citrus .
dbr:Citrus rdfs:label "Citrus"@en ;
  dbo:family dbr:Rutaceae ;
  owl:sameAs fb:en.citrus .
```

The last triple establishes an `owl:sameAs` *relation to another RDF resource referring to citrus, published by an external exporter of RDF (Freebase). On the Freebase site, we can find the following (subset of) information about citrus:*

```
# PREFIXES ...

fb:en.citrus fb:scientific_name "Citrus"@en ;
  fb:higher_classification fb:en.rutaceae ;
  fb:lower_classifications fb:en.lemon ;
  fb:lower_classifications fb:en.madarin_orange ;
  fb:lower_classifications fb:en.pomelo .
fb:en.pomelo fb:higher_classification fb:en.citrus .
```

We see two independent exporters on the Web publishing RDF data about citrus using two different URIs. However, the `owl:sameAs` *link provided by DBpedia states that the two citrus URIs refer to the same thing. Hence, information published about citrus in RDF under one of the URIs also applies to the other URI (called the principle of replacement). This feature of OWL is axiomatized by rules* `eq-*` *in Table 1.2. Applying these rules, one can see that taken together, the above two graphs entail:*

```
# PREFIXES ...

fb:en.citrus owl:sameAs dbr:Citrus .
dbr:Lemon dbo:genus fb:en.citrus .
fb:en.citrus rdfs:label "Citrus"@en ;
  dbo:family dbr:Rutaceae ;
  owl:sameAs fb:en.citrus .
dbr:Citrus fb:scientific_name "Citrus"@en ;
  fb:higher_classification fb:en.rutaceae ;
  fb:lower_classifications fb:en.lemon ;
  fb:lower_classifications fb:en.madarin_orange ;
  fb:lower_classifications fb:en.pomelo .
fb:en.pomelo fb:higher_classification dbr:Citrus .
```

Thus, the semantics of `owl:sameAs` *can be used to link and combine RDF data about the same resources from multiple locations.*

Unlike RDFS, no finite set of rules can support (either of) the complete semantics of OWL; hence this ruleset and any such ruleset can only partially axiomatize (i.e., encode) the semantics of the highlighted subset of features.

On top of all of these features—and a lot more besides those that we have introduced—OWL defines two standard and compatible semantics. The first semantics is called the "RDF-Based Semantics", which is defined for any RDF data and is backwards-compatible with RDF [53]. However, all typical reasoning tasks over an OWL (2) Full ontology—such as consistency checking, satisfiability checking (checking if a class can have a member without causing a logical contradiction), subsumption checking (checking if a class is necessarily a sub-class of another), instance checking (checking if a resource is a member of a class) and conjunctive query answering (posing complex queries against the ontology and its entailments)—are *undecidable*. This means that such automated tasks cannot be guaranteed to ever terminate for ontologies described in the unrestricted OWL Full language.

The second semantics, called the "Direct Semantics", can only interpret OWL ontologies that abide by certain restrictions. These restrictions are such that the ontologies described by the language can be translated into *axioms* compatible with a formalism called Description Logics (DL). The core aim of DL is to define a subset of First Order Logic (FOL) for which certain reasoning tasks are known to be decidable, where the semantics of the language can be supported in a sound (correct) and complete manner using known algorithms. However, we already mentioned that inference rules (of the style we have already introduced) are insufficient to support the semantics of OWL in a complete manner, where other inference mechanisms are sometimes used. One popular algorithm in this area is the use of specialized tableau-based approaches inherited from FOL [52]. Thus, unlike the unrestricted RDF-Based Semantics, ontologies that conform to the restrictions laid out by the Direct Semantics have known algorithms for sound and complete reasoning. These guarantees of sound and complete reasoning are often important in critical applications, where the possibility of an incorrect or missing answer would not be acceptable. This introduces a core trade-off of expressivity of the language (features it supports), versus the efficiency of complete reasoning tasks over that language. The OWL standards thus define (sub-)languages that can be interpreted under Direct Semantics and that explore this trade-off.

OWL 1 DL was defined by the original OWL standard to be the maximal language for which the original version of the Direct Semantics is defined. For tasks such as consistency checking, satisfiability checking, classification, etc., OWL (1) DL is NExpTime-complete [24] with respect to the number of input axioms, which is a very high worst-case complexity. Conjunctive query answering is not yet known to be (un)decidable for OWL 1 DL.[13]

OWL 1 Lite was also defined by the original OWL standard, and aimed to restrict the use of problematic OWL 1 DL features so as to arrive at a more "efficient" OWL sub-language. That said, OWL 1 Lite is ExpTime-complete [24] with respect to input axioms for the previously mentioned tasks: still a very high complexity.

OWL 2 DL was defined by the OWL 2 standard, and is the maximal language for which the updated Direct Semantics of OWL (2) are defined, adding a lot of new features above and beyond OWL 1 DL. The analogous complexity of OWL 2 DL is 2NExpTime-complete [24]: an extremely high worst-case complexity. Conjunctive query answering is not yet known to be (un)decidable for OWL 2 DL.

OWL 2 EL was the first *OWL profile* to be defined by the OWL 2 standard. OWL profiles are syntactic subsets of OWL 2 DL for which polynomial-time algorithms are known for various reasoning tasks. The OWL 2 EL profile targets support for expressive class axioms, disallowing the use of certain property axioms: OWL 2 EL is PTime-complete (deterministic polynomial complexity) for all reasoning tasks except conjunctive

[13] Glimm and Rudolph [21] have proven decidability for conjunctive query entailment with respect to the Description Logic underlying OWL DL, but under the assumption that transitive properties (or properties that entail transitive properties) do not appear as predicates in the query. (They believe that the result extends to OWL 2 DL; they do not currently address a complexity bound.)

query-answering, for which it is PSPACE-complete [24] in a combined complexity (with respect to assertions, axioms and query size).

OWL 2 QL is the second OWL profile and is aimed at “query-rewriting” implementations over relational database systems, such that structured queries are expanded to request asserted data that may entail some subgoal of the query. The aforementioned reasoning tasks are NLOGSPACE-complete with the exception of conjunctive query answering which is NP-complete (combined complexity).

OWL 2 RL is the third OWL profile and is designed to be supported by (in particular) rule-based inferencing engines. It is based on previous proposals to partially support OWL semantics using rules, such as Description Logic Programs (DLP) [25] and pD* [58]. Along these lines, OWL 2 RL is a syntactic subset of OWL 2 DL with an accompanying set of OWL 2 RL/RDF entailment rules (some of which are presented in Table 1.2) such that the entailments possible for OWL 2 RL though Direct Semantics (often implemented using tableau-based approaches) are partially aligned with the entailments given by the OWL 2 RL/RDF rules with respect to the RDF-Based Semantics. OWL 2 RL is PTIME-complete for all reasoning tasks except conjunctive query-answering, for which it is NP-complete [24].

Again, OWL is a complex standard and full details are out of scope. For more information about Description Logics and the underlying formal aspects of OWL, we refer the interested reader to Baader et al.’s “Description Logic Handbook” [3] and also to more recent and more succinct primers by Rudolph [51] and by Krötzsch [38].

It is sufficient for the purposes of this book to understand that OWL goes far beyond RDFS and brings a much richer semantics for use with RDF data, and to have a basic understanding of the semantics for the subset of OWL features introduced. Various restricted sub-languages of OWL are defined that allow for guarantees of sound and complete reasoning algorithms, with a variety of different computational complexities. Furthermore, OWL can be supported by rules such as those enumerated in Table 1.2, but such rules can only support a subset of the semantics of OWL in a complete manner. One well-defined subset of OWL (2) for which rules are “enough” is the OWL 2 RL profile, for which a ruleset called OWL 2 RL/RDF is defined. The OWL 2 RL/RDF ruleset (and its subsets) can be applied over ontologies that fall outside of OWL 2 RL, and can, for example, provide sound but incomplete reasoning over OWL 2 Full; this is appealing since OWL 2 RL/RDF rules can thus be applied directly over arbitrary RDF datasets to derive inferences.

1.5 Querying RDF with SPARQL

The SPARQL standard centers around a query language designed specifically for RDF data [50], as well as a protocol by which SPARQL queries can be invoked and their results returned over the Web [15]. The original SPARQL specification became a W3C Recommendation in 2008 [50]. In 2013, SPARQL 1.1—an extension of the original SPARQL standard—also received W3C Recommendation [27]. Herein, we focus primarily on the features of the original SPARQL standard.

SPARQL itself is orthogonal to the RDFS and OWL standards outlined previously, and is built directly on top of the RDF data-model without direct support for inferencing (cf. Figure 1.1).[14] SPARQL is similar in respects to the Structured Query Language (SQL) used for querying relational databases, sharing certain query features and keywords (but in the case of SPARQL, designed for interacting with RDF data). The RDF-specific syntax of SPARQL is closely tied with that of Turtle: familiarity with Turtle syntax will greatly help in understanding SPARQL syntax.

On a high level, a SPARQL query can consist of up to five main parts:

Prefix Declarations allow for defining URI prefixes that can be used for shortcuts later in the query (in a similar fashion to Turtle).

Dataset Clause allows for specifying a closed partition of the indexed dataset over which the query should be executed.

Result Clause allows for specifying what type of SPARQL query is being executed, and (if applicable) what results should be returned.

Query Clause allows for specifying the query patterns that are matched against the data and used to generate variable bindings.

Solution Modifiers allow for ordering, slicing and paginating the results.

Example 10. *We give a brief example of a SPARQL query containing each of the above five parts. Comment lines are prefixed with '*`#`*'. The query first defines prefixes that can be re-used later in a similar fashion to that allowed by Turtle (although prefixes are not terminated with periods in SPARQL). Next the* `# DATASET CLAUSE` *selects partitions of the dataset over which the query should be run: in this case, an RDF document on the DBpedia site about lemons. Thereafter, the* `# RESULT CLAUSE` *states what kind of results should be returned for the query: in this case, a unique (i.e.,* `DISTINCT`*) set of pairs of*

[14] Integration of RDFS and OWL entailment with SPARQL has recently been standardised alongside SPARQL 1.1 in the form of "SPARQL 1.1 Entailment Regimes" [20]. However, entailment regimes are not part of the core of the SPARQL 1.1 query language and are instead an optional standard that can be layered on top.

RDF terms matching the `?genus` *and* `?order` *variables respectively. Next, the* `# QUERY CLAUSE` *states the patterns that the query should match against: in this case, looking up the values for the* `dbo:genus` *and* `dbo:order` *of* `dbr:Lemon`*. Finally, the* `# SOLUTION MODIFIER` *section allows for putting a limit on the number of results returned, to order results, or to paginate results: in this case, a maximum (i.e.,* `LIMIT`*) of two results is requested from the query.*

```
# PREFIX DECLARATIONS
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
# DATASET CLAUSE
FROM <http://dbpedia.org/data/Lemon.xml>
# RESULT CLAUSE
SELECT DISTINCT ?genus ?order
# QUERY CLAUSE
WHERE {
  dbr:Lemon dbo:genus ?genus ;
     dbo:order ?order .
}
# SOLUTION MODIFIER
LIMIT 2
```

If one were to assume that the triples:

```
dbr:Lemon dbo:genus dbr:Citrus ;
   dbo:order dbr:Rosids , dbr:Sapindales .
```

were present in the graph `<http://dbpedia.org/data/Lemon.xml>` *indexed by the store, we would expect a result like:*

?genus	?family
dbr:Citrus	dbr:Sapindales
dbr:Citrus	dbr:Rosids

where the header indicates the variables for which the respective terms are bound in the results, based on the `SELECT` *clause of the query.*

We now detail the function of the latter four parts of a SPARQL query. Due to the breadth of the SPARQL language, we do not provide examples for the features introduced, but instead refer the interested reader to the official SPARQL documentation where many such examples are introduced [50]. Instead, we give an overview of the different features and outline a common formalization for SPARQL queries (an extension of that originally proposed by Pérez et al. [46]), which will be re-used in later chapters of the book.

1.5.1 Query Types

A SPARQL query can be one of four types:

`SELECT [DISTINCT|REDUCED]`: Requests a list of bindings for variables specified by a query. By default, the `SELECT` query will return duplicate results corresponding directly to the number of unique (RDF) graph patterns matched by the query in the data. The optional `DISTINCT` keyword will enforce unique tuples, removing duplicate results. Another alternative is to use the `REDUCED` keyword, which states that duplicates are allowed but do not need to have the same number of results as per the default semantics of `SELECT`, allowing the query engine to optimize accordingly (and to avoid having to execute a uniqueness check).

`ASK`: Returns a boolean value indicating whether or not there was a match in the data for the query clause. An `ASK` query is *roughly* the same as running a `SELECT` query and seeing if there are non-empty results.

`CONSTRUCT`: Provides an RDF template into which variables bound in the query clause can be inserted. When the query is executed, for each result tuple, variables in the template are bound accordingly generating (ground) RDF as a result for the query.

`DESCRIBE`: Asks the endpoint to provide an RDF description for a particular RDF term. `DESCRIBE` can also be used to describe bindings to a variable, where each description is added (by set union) to the query output. The nature of the description returned for each term is not specified in the SPARQL standard, and is thus left to the discretion of developers and administrators. Informally, common types of `DESCRIBE` functionalities implemented in practice are:

- return all triples where the term is mentioned in either (i) any position, (ii) as a subject, or (iii) as a subject or object.
- return some form of Concise Bounded Descriptions (CBD) (please see [57] for details).

In the above four types of SPARQL queries, there is an important distinction to make: the first two return solutions (not RDF), and the latter two return RDF. RDF can be returned in any appropriate syntax, though RDF/XML is required for compliance with the standard. Results for `SELECT` and `ASK` queries are typically serialized in a custom XML syntax defined in the SPARQL standard [50, § 10], though other result formats based on JSON, CSV, TSV, etc., are also common.

The result clause of a SPARQL query is the only mandatory part of a SPARQL query, where "`DESCRIBE <`*`someuri`*`>`" by itself is a valid SPARQL query. However, almost all queries will also contain (at least) a query clause.

1.5.2 Dataset Clause and Named Graphs

In Example 10, we hinted at the fact that SPARQL queries are executed over "partitioned" datasets of RDF, and not over a single monolithic RDF graph: SPARQL operates over a *SPARQL dataset* which is composed primarily of *named graphs*.

Definition 5. *A SPARQL* named graph *is defined as a pair* (u, G) *where* u *is a URI serving as a name for the graph* ($u \in \mathbf{U}$)[15] *and* G *is an RDF graph (per Definition 3).*

A SPARQL dataset is then composed of a *default graph*, which is an unnamed RDF graph, and a set of named graphs.

Definition 6. *A* SPARQL dataset $D = \{G_D, (u_1, G_1), \ldots, (u_n, G_n)\}$ *is a set of (named) graphs where* $u_1 \ldots u_n$ *are distinct URIs and* $G_D, G_1, \ldots G_n$ *are RDF graphs. The unnamed graph* G_D *is called the default graph. Each pair* (u_i, G_i) *is a named graph. We use the notation* $D(u_i) = G_i$ *to select a graph from a dataset based on its name.*

A SPARQL dataset is thus composed of a set of RDF graphs that are named, and a default graph. This allows for querying configurable partitions of a SPARQL dataset in isolation, using the available URI names to "load" individual graphs for querying, creating a query-specific SPARQL dataset containing a custom default graph and a selection of named graphs from those accessible by the query engine. This selection of a dataset is done in the dataset clause of the query, which can be specified using two optional features:

`FROM`**:** The `FROM` keyword is used to define the default graph for a query-specific SPARQL dataset. Each `FROM` keyword is used in combination with the URI of a named graph. All URIs loaded in this manner will be added to the default graph for the query, using an RDF merge (see Definition 4) to ensure that blank node labels are kept distinct between source graphs.

`FROM NAMED`**:** The `FROM NAMED` key-phrase is used to define the set of named graphs that are added to a query-specific SPARQL dataset. Each `FROM NAMED` key-phrase is used in combination with the URI of a named graph, which will be added to the query dataset.

The default graph is the graph against which patterns in the query clause are matched if no `GRAPH` clause is explicitly mentioned (referred to later). For queries without a dataset clause, many SPARQL engines implement a full default query dataset, containing a default graph with all data that they index

[15] Strictly speaking, SPARQL is based on IRIs, not URIs, and named graphs use IRIs; recall from earlier in the chapter that we simplify discussion by referring analogously to IRIs as URIs.

(either the RDF merge or more often the set union of all graphs[16]), and all named graphs loaded: queries without a dataset clause will thus often be run over all known data. Hence, explicit dataset clauses in queries are typically used to *restrict* the dataset by specifying only those graphs over which the query should be run.[17] A minor issue of note: when `FROM` and/or `FROM NAMED` are used, the default graph for the query dataset is initialized as empty.

1.5.3 Query Clause

The query clause is undeniably where the magic happens in a SPARQL query: based on the selected dataset, it specifies the query patterns and other criteria that query variables must match to be returned to the other parts of the query. In this section, we lay out the core features of a query clause, and formalize these notions following conventions laid out by Pérez et. al [46] and the SPARQL standard [50, § 12].

The query clause is (almost always) announced using the `WHERE` keyword, and is surrounded by opening and closing braces. The core of a query clause is often one or more sets of *triple patterns*, where each set is called a *basic graph pattern* (BGP). An example of a basic graph pattern was provided in Example 10 with two triple patterns embedded in a `WHERE` clause.

Definition 7. *Let* $\mathbf{V}$ *denote a set of variables that range over all RDF terms. An* RDF triple pattern *tp is an RDF triple where query variables are allowed in any position:* $tp \in (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$.[18] *A set of triple patterns is called a* basic graph pattern *(BGP).*

In RDF triple patterns, blank nodes are considered as existential variables, meaning that they will not match blank-nodes with the same label, but rather function as query variables whose bindings cannot be used outside of the query clause (and thus that cannot be returned in results). Henceforth, we do not treat blank nodes in SPARQL query patterns since they can be treated analogously to *non-distinguished query variables*: variables that cannot be used elsewhere outside of the query-clause scope.

Triple patterns can then be executed against RDF graphs to produce a set of solution mappings[19], such that each mapping applied to the triple pattern returns a triple present in the RDF graph.

Definition 8. *Let* μ *be a* solution mapping *from a set of variables to RDF terms:* $V \to \mathbf{U} \cup \mathbf{L} \cup \mathbf{B}$ *where* $V \subset \mathbf{V}$. *The set of variables* V *for which* μ *is defined is called the* domain *of* μ, *termed* $dom(\mu)$. *We abuse notation and*

[16] See `http://www.w3.org/TR/sparql11-service-description/#sd-uniondefaultgraph`

[17] *Negation* of graphs is only possible in the query clause, using a combination of `GRAPH` and `FILTER`.

[18] SPARQL triple patterns allow literals in the subject to future-proof for a future version of RDF where such is allowed (no such plans yet exist).

[19] SPARQL rather defines a *sequence* of solution mappings since ordering can be important when solution modifiers are taken into account. However, we leave this implicit for brevity.

allow $\mu(tp)$ to denote a solution mapping for a triple pattern such that $\mu(tp)$ uses μ to replace all variables in tp with RDF terms.

Definition 9. *Let G be an RDF graph, let tp be a triple pattern, and let $vars(tp)$ denote the set of variables in tp. We denote by:*

$$[\![tp]\!]_G = \{\mu \mid \mu(tp) \in G \text{ and } dom(\mu) = vars(tp)\}$$

the execution of the triple pattern tp against G, which returns the set of all mappings (with minimal domains) that can map tp to a triple contained in G.

A basic graph pattern can then comprise of multiple such triple patterns, considered as a conjunction: during execution, this conjunction leads to a *join operation* over *compatible* mappings present in the sets of mappings produced for individual triple patterns.

Definition 10. *Two solution mappings μ_1 and μ_2 are termed* compatible*—which we denote by $\mu_1 \sim \mu_2$—if and only if both mappings correspond for the variables in their overlapping domain. More formally, $\mu_1 \sim \mu_2$ holds if and only if for all variables $v \in dom(\mu_1) \cap dom(\mu_2)$, it holds that $\mu_1(v) = \mu_2(v)$. If $\mu_1 \sim \mu_2$, then $\mu_1 \cup \mu_2$ remains a valid mapping. Let M_1 and M_2 be a set of mappings. We define a* join *over two sets of mappings as:*

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2 \text{ such that } \mu_1 \sim \mu_2\}$$

In other words, for each pair of compatible mappings μ_1 and μ_2 from the corresponding sets, the join operation adds a new mapping by extending the bindings of μ_1 with those from μ_2 (or equivalently, vice-versa).

Definition 11. *Let $B = \{tp_1, ..., tp_n\}$ be a basic graph pattern. The execution of B for a graph G is given as:*

$$[\![B]\!]_G = [\![tp_1]\!]_G \bowtie \ldots \bowtie [\![tp_n]\!]_G$$

where joins are commutative and associative (and thus can be executed in any order). If we further abuse notation and allow $\mu(B)$ to denote the application of μ to all variables in B, we can alternatively state this as:

$$[\![B]\!]_G = \{\mu \mid \mu(B) \subseteq G \text{ and } dom(\mu) = vars(B)\}$$

which is analogous to Definition 9, but applied for basic graph patterns.

On top of Basic Graph Patterns, SPARQL defines four other core features that can be used to create complex *query patterns* in query clauses:

`GRAPH`: When used with a URI, `GRAPH` specifies the named graph (from the query dataset) against which a BGP should be matched. When used with a variable, `GRAPH` can be used to bind (or join) the named graph for which a BGP is matched. A BGP with a surrounding `GRAPH` clause cannot access the default graph for the query dataset, only its named graphs.

UNION: Allows for defining a disjunction of query patterns that the query should match. The result is the union of the sets of solution mappings generated for each disjunctive query pattern.

OPTIONAL: Allows for specifying optional query patterns that the query should try to match. If nothing is matched, instead of applying a conjunction and removing the solution during execution, variables unique to the optional pattern are mapped to UNBOUND.

FILTER: Can be used to specify further conditions that a query solution should match. Conditions can comprise of various operators, built-in functions, casting operations and boolean connectives, resulting in an expression that takes a solution mapping as input and returns `true` or `false`. If the expression evaluates to `false`, the solution mapping is filtered.

- **Operators** include equality and inequality operators for RDF terms (including less/greater than for range queries, etc.).
- **Built-in functions** can be used to transform and examine RDF terms, including testing the type of an RDF term (is it a URI, blank-node, literal or is it unbound), parsing functions for literals (e.g., return language tags or datatype URIs), regex functions, and so forth.
- **Casting operations** allow for converting between different types of datatype literals, or converting URIs to a string, etc.
- **Boolean connectives** include conjunction (`&&`), disjunction (||) and negation (`!`), allowing to combine sub-expressions.
- **User-defined Functions** allow for custom built-ins to be defined by vendors if not provided by SPARQL.

As stated at the outset, these features can combine to create a complex SPARQL query pattern.

Definition 12. *A query pattern can be defined recursively as:*

- *Any basic graph pattern is a query pattern P.*
- *If P is a query pattern and x is a URI or a variable, then selecting a graph (P GRAPH x) is also a query pattern.*
- *If P_1 and P_2 are query patterns, then their combination through conjunction (P_1 AND P_2), union (P_1 UNION P_2) or optional (P_1 OPT P_2) is also a query pattern.*
- *If P is a graph pattern and R is a filter condition as described above, then (P FILTER R) is also a query pattern. Henceforth, we refer to R as a function that maps from a solution mapping to a boolean value.*

Beyond (inner) joins for conjunctions (Definition 10), the execution of a query pattern requires one more non-trivial operator over sets of solutions: *left-join* (for `OPTIONAL`).

Definition 13. *Let M_1 and M_2 be a set of mappings. Let $M_1 - M_2$ denote the set of all mappings in M_1 that have no compatible mapping in M_2. Then, the* left-join *of M_1 and M_2 is defined as:*

$$M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 - M_2)$$

Compatible mappings between M_1 and M_2 are joined (per Definition 10) and added to the result. Additionally, mappings in M_1 without a compatible mapping in M_2 are also preserved in the result (for these latter mappings, variables unique to M_2 are undefined by the map and result in `UNBOUND`*).*

The execution of a SPARQL query pattern can then be defined for a SPARQL dataset as follows:

Definition 14. *Let D be a SPARQL dataset. Let P be a SPARQL query pattern, per Definition 12. Let G intuitively denote the* active graph *of D. We can define the execution of P over D for the active graph G, denoted ${}_D[\![P]\!]_G$, recursively as follows:*

- *if P is a basic graph pattern, then ${}_D[\![P]\!]_G = [\![P]\!]_G$ per Definition 11;*
- *else if P has the form ...*

  ... P_1 `AND` P_2, then ${}_D[\![P]\!]_G = {}_D[\![P_1]\!]_G \bowtie {}_D[\![P_2]\!]_G$;
  ... P_1 `UNION` P_2, then ${}_D[\![P]\!]_G = {}_D[\![P_1]\!]_G \cup {}_D[\![P_2]\!]_G$;
  ... P_1 `OPT` P_2, then ${}_D[\![P]\!]_G = {}_D[\![P_1]\!]_G ⟕ {}_D[\![P_2]\!]_G$;
  ... P' `FILTER` R, then ${}_D[\![P]\!]_G = \{\mu \in {}_D[\![P']\!]_G \mid R(\mu) = \texttt{true}\}$;

- *else if P has the form P' `GRAPH` x, and if ...*

  ... $x \in \mathbf{U}$, then ${}_D[\![P]\!]_G = {}_D[\![P']\!]_{D(x)}$;
  ... $x \in \mathbf{V}$, then

$$ {}_D[\![P]\!]_G = \bigcup_{(u_i,G_i)\in D} {}_D[\![P']\!]_{G_i} \bowtie \left\{\{(x,u_i)\}\right\}.$$

Slightly abusing notation, we denote by the shortcut $[\![P]\!]_D = {}_D[\![P]\!]_{G_D}$ the execution of a query pattern P against the dataset D. *This involves initially setting the default graph of the dataset G_D as the active graph.*

A SPARQL query clause is then composed of a query pattern. Collectively, these definitions provide a semantics for SPARQL query clauses and how they are executed over a dataset defined by the dataset clause. The solution mappings from the query clause can then be further chopped and changed through solution modifiers (discussed next) before serving as input for the query type projection/serialization (`SELECT`, `CONSTRUCT`, etc.).

1.5.4 Solution Modifiers

For the `SELECT`, `CONSTRUCT` and `DESCRIBE` SPARQL query types, further options are available to post-process results produced from the query clause using *solution modifiers*. Such options are redundant for `ASK` queries, which only return a single `true`|`false` result. For the moment, we focus on the use of solution modifiers for `SELECT` queries, which return a list of solution mappings; later, we will mention solution modifiers in the context of `CONSTRUCT` and `DESCRIBE` queries, which return RDF.

The following solution modifiers are available for use in SPARQL:

`ORDER BY [ASC|DESC]`: The `ORDER BY` clause assigns a list of variables by which to sort results using SPARQL's natural ordering over (most) RDF terms [50, § 9.1].[20] Sorting is performed lexicographically based on variable order. The optional `ASC` and `DESC` keywords specify whether sorting for a specific variable should be in ascending or descending order, with the former being the default.

`LIMIT`: The `LIMIT` clause allows for specifying a non-negative integer n, where n specifies the maximum number of results to return.

`OFFSET`: The `OFFSET` clause takes a non-negative integer n and tells the SPARQL engine to skip over the first n results that would be returned. In combination with `LIMIT`, this allows for a form of pagination of results. However, strictly speaking, `OFFSET` is only useful in combination with `ORDER BY` since no default ordering is specified (and furthermore, the default ordering can be non-deterministic). Hence, if asking for results 1–10 in a first query and subsequently asking for results 11-20 in a second query, without `ORDER BY` there are no guarantees that these results will be "sequential" or even "disjoint" in the intuitive sense.

The above discussion applies directly to `SELECT` queries. For `CONSTRUCT` and `DESCRIBE` queries, solution modifiers are used to select the list of solutions from which RDF will be *subsequently* generated; note that the `ORDER BY` clause on its own has no meaningful effect for such queries since the output will be an unordered set of RDF triples: however, in combination with `LIMIT` and `OFFSET`, `ORDER BY` can be used to select a deterministic subset of solutions with which to generate RDF results.

1.5.5 Towards SPARQL 1.1

We have seen that SPARQL is quite an expressive query language for RDF, offering a wide range of features including four different query types;

[20] For example, no ordering is defined between two literals with language tags. This may lead to implementation-specific side-effects in sorting while not affecting SPARQL compliance.

methods to chop and select query-specific datasets using named graph mechanisms; a variety of query-pattern features including joins, optional pattern matching, disjunctive union patterns, and various filter operators; as well as solution modifiers to shape the final results. Going even further, SPARQL 1.1 was recent standardised [27] and extends SPARQL with a wide-range of new features. We briefly summarize some of the main novelties.

In terms of the core query language, the following novel features have been introduced:

Property Paths allow for specifying chains of non-fixed paths in a query clause using a limited form of regular expressions over RDF predicates.

Aggregates allow for grouping results in such a manner that functions like max, min, sum, avg, count, etc., can be applied over solutions grouped by common terms bound to a given set of variables.

Binding Variables enables initializing new variables to, e.g., hold the result from the execution of a function, or a set of constant terms, etc.

Subqueries allow for nesting sub-`SELECT` queries, where nested queries are executed first and the results projected to the outer query. Most importantly, subqueries allows for non-query clause operators (in particular, solution modifiers for ordering, limiting and offseting) to be used within a query clause.

Aside from extensions to the core query language, a few other broader extensions to the SPARQL standard are also currently on the table:

Entailment [20]: SPARQL does not leverage RDF(S) or OWL Semantics when running queries. The SPARQL 1.1 Entailment Regimes proposal aims to offer optional support for such semantics when running SPARQL queries, allowing to find additional answers through formal entailment mechanisms.

Update [18]: In the original SPARQL specification, there was no standardized method by which the content of a SPARQL dataset could be updated. SPARQL 1.1 Update aims to rectify this by providing an update language (similar in many respects to the query language itself) that allows for modifying the content of the index, possibly based on the results of a nested query clause.

Federation [49]: SPARQL federation involves executing a single query over a selection of SPARQL endpoints. SPARQL 1.1 Federated Query centers around the `SERVICE` clause, which can be nested in a SPARQL query clause and allows for invoking a sub-query against a remote endpoint at a given URL.

Service Descriptions [60]: To allow for automated discovery of SPARQL endpoints on the Web and their inherent functionalities, the SPARQL 1.1 Service Descriptions proposal provides a vocabulary to comprehensively describe the features supported by that endpoint. The Service Description of an endpoint can be retrieved by performing a HTTP lookup against the endpoint URL requesting content in a suitable RDF format.

CSV, TSV and JSON outputs [54, 55]: New formats for outputting results to SPARQL `SELECT` and `ASK` queries have been formalized, allowing for easier integration of SPARQL engines into software applications (by, e.g., not requiring XML parsers).

SPARQL 1.1 thus represents a significant expansion of the SPARQL set of standards.

1.6 Linked Data

Thus far we have discussed four of the core Semantic Web standards: RDF, RDF Schema, the Web Ontology Language, and SPARQL. However, aside from dropping the occasional reference to things like "URIs" and "URLs", and a brief mention of SPARQL protocol, we have spoken very little about the Web itself. This is mainly because these four standards can be considered as "languages", and much like one could discuss HTML without mentioning HTTP and the Web (except for mentions of the URIs used to link to remote documents and images), one can discuss RDF and RDFS and OWL and SPARQL without mentioning HTTP and the Web (except for mentioning the URIs used to name things and, where applicable, assuming an Open World). Thus, it may be surprising to learn that the core Semantic Web standards themselves say precious little about the Web, except for using URIs.

The core aim of Linked Data then, is to provide a set of principles by which the Semantic Web standards can be effectively deployed on the Web in a manner that facilitates discovery and interoperability of structured data. To understand how the core tenets of Linked Data came about, we first give a little background leading up to the proposal for the Linked Data principles.

1.6.1 The Early Semantic Web on the Web

With regards to publishing RDF on the Web, early efforts produced large, insular "‘data silos", often a dump of potentially huge RDF documents. Such silos included OpenCyc comprising of axioms specifying general knowledge[21],

[21] `http://www.cyc.com/2004/06/04/cyc`; retr. 2012/11/30

the GALEN ontology for describing medical terminology[22], exports from UniProt describing protein sequences[23], and exports from the WordNet lexical database[24]. Although such dumps have their own inherent value and are published in an interoperable data-model through RDF, they rarely interlink with remote data (if at all) and they are published using different conventions (e.g., in a Web folder, using different archiving methods, etc.) thus making them difficult to discover automatically. Effectively, such dumps are isolated islands of data that only use the Web for file transfer.

One notable exception to the emerging RDF silos was the publishing centered around the Friend Of A Friend (FOAF) community. In early 2000, Brickley and Miller started the "RDF Web Ring" project (or simply "RDFWeb") which was eventually rebranded as FOAF.[25] FOAF provides a lightweight vocabulary containing various terms for describing people; initially comprising of a set of RDF properties, this vocabulary evolved throughout the years to include RDFS and OWL descriptions of properties *and* classes, continuously adapting to community feedback and requirements. Various tools solidified adoption of the FOAF vocabulary, including the 2004 FOAF-a-Matic generator, which allowed users to fill some personal details into a form to generate a FOAF profile (an RDF document on the Web describing them) and to link to the FOAF profiles of their friends. FOAF profiles were deployed by Semantic Web enthusiasts, with more adventurous adopters creating ad-hoc vocabularies to extend the personal details contained within, and so an early *Web of Data*—a Web of Semantic Web documents—formed [16].

However, early FOAF data—and RDF data in general—made sparse use of URIs, preferring to use blank nodes.[26] Thus, consistent URI naming across documents was not even attempted, and there was no *direct* means of finding information about a given resource. As more and more RDF data were published on the Web and more and more RDFS and OWL vocabularies became available, there was an eminent need in the community for a set of best practices. The first notable step in this direction was the publication in March 2006 of a W3C Working Note entitled "Best Practice Recipes for Publishing RDF Vocabularies" [44], which described URI naming schemes for vocabulary terms, and the HTTP mechanisms that should be used to return information upon lookup of those URIs (called *dereferencing*). These best practices aligned with the then recent Web Architecture W3C Recommendation [34] and with practices already used by, for example, the FOAF vocabulary. Similar rec-

[22] `http://www.co-ode.org/galen/full-galen.owl`; retr. 2012/11/30

[23] `http://www.uniprot.org/`; retr. 2010/11/30

[24] `http://lists.w3.org/Archives/Public/www-rdf-interest/2001Feb/0010.html`; dataset now offline

[25] `http://www.foaf-project.org/original-intro`; retr. 2010/11/02

[26] With a couple of exceptions: for example, in 1999, Brickley had already begun experiments with respect to publishing WordNet as RDF on the Web in a manner analogous to modern Linked Data. See discussion at `http://lists.w3.org/Archives/Public/www-rdf-interest/2001Feb/0010.html`; retr. 2013/06/14 (and with thanks to the anonymous reviewer who pointed this out).

ommendations were then generalized and broadened for arbitrary RDF data, leading to the Linked Data principles and best practices.

1.6.2 Linked Data Principles and Best Practices

In July 2006, Berners-Lee published the initial W3C Design Issues document outlining Linked Data principles, rationale and some examples [7]. This document generalized the earlier best-practices for vocabularies, similarly espousing use of dereferenceable HTTP URIs for naming, and additionally encouraging inclusion of external URIs as a simple form of linking. The four Linked Data principles are as follows (paraphrasing Berners-Lee [7]):

1. *use URIs* as names for things;
2. *use HTTP URIs* so those names can be looked up (aka. *dereferencing*);
3. *return useful information* upon lookup of those URIs (esp. RDF);
4. *include links* by using URIs which dereference to remote documents.

The result can be conceptualized as a Web of Data, where URIs identify *things*, dereferencing URIs (through HTTP) returns structured data (RDF) about those things, and that structured information is inherently composed of related URIs that constitute links to other sources enabling further discovery.[27]

The central novelty of Linked Data when compared with traditional Semantic Web publishing was the emphasis on using dereferenceable URIs to name things in RDF. With data published under the Linked Data principles, to find out more information about the resource identified by a particular URI, you could look it up through HTTP using content-negotiation methods requesting RDF data and expect to find a document describing that resource.

Example 11. *The data used in our running example thus far is sourced from a Linked Data exporter called DBpedia. The URLs of the documents from which these RDF data are retrieved are provided as comments (prefixes are omitted for brevity):*

```
# http://dbpedia.org/data/Lemon.xml

dbr:Lemon rdfs:label "Lemon"@en ;
   dbp:calciumMg 26 ;
   dbo:genus  dbr:Citrus  ;
   ...
```

```
# http://dbpedia.org/data/Citrus.xml

dbr:Citrus rdfs:label "Citrus"@en ;
   dbo:family dbr:Rutaceae ,
      dbr:Aurantioideae ;
   ...
```

[27] We note that the term “Web of Data” is a contentious one. In particular, opinions differ on whether RDF and Semantic Web technologies are integral to such a Web of Data, or are simply one direction to pursue. However, the phrase “web of data” was used in the explicit context of the Semantic Web as early as 1998 by Berners-Lee [6].

An agent encountering the document about `dbr:Lemon` *on the left will discover that its genus is* `dbr:Citrus`*. However, no further information about the resource identified by* `dbr:Citrus` *is available in that document. Instead, by looking up the URI using appropriate content-negotiation (dereferencing), the agent can retrieve the document on the right, discovering an RDF description of the resource identified by* `dbr:Citrus`*. Analogously, looking up* `dbr:Lemon` *will return the document on the left. Other URIs such as* `rdfs:label`, `dbp:calciumMg`, `dbr:Rutaceae`*, etc. can also be looked up to retrieve RDF descriptions, thus providing links to other remote documents.*

The core message of the Linked Data community is, in essence, a bottom-up approach to bootstrapping Semantic Web publishing, where the emphasis is not on languages, but rather on publishing and linking structured data on the Web in a manner that facilitates interoperability. This bottom-up philosophy is best epitomized by the (Linked) Open Data "5 Star Scheme" [7], summarized as follows:

⋆	Publish data on the Web under an open license
⋆ ⋆	Publish structured data
⋆ ⋆ ⋆	Use non-proprietary formats
⋆ ⋆ ⋆ ⋆	Use URIs to identify things
⋆ ⋆ ⋆ ⋆ ⋆	Link your data to other data

Here, each additional star is promoted as increasing the potential reusability and interoperability of the publishers' data. Although the final star does not explicitly mention publishing through RDF, its use is (often understood to be) implied given a lack of viable alternative structured formats where URIs are used as identifiers and links can thus be embedded in the content.

1.6.3 Linking Open Data

Promoting Linked Data principles, the W3C announced a new Community Project called "Interlinking Open Data"—subsequently shortened to "Linking Open Data" (LOD)—inspired by the growth in *Open Data* published on the Web under liberal licenses. The goal of the Linked Open Data project is twofold: (i) to introduce the benefits of RDF and Semantic Web technologies to the Open Data movement, (ii) to bootstrap the Web of Data by creating, publishing and interlinking RDF exports from these open datasets [30].

Furthermore, the community set about developing and promoting an extended set of Linked Data publishing guidelines on top of the core principles outlined previously. The full set of publishing guidelines are out of scope, where we instead refer the interested reader to Heath and Bizer's recent book on the topic [30]. Herein, we summarize some of the main guidelines:

Dereferencing practices: In RDF, URIs can be used to identify anything, not just documents. Linked Data guidelines thus recommend a level of indirection to signify this distinction on a HTTP level, using either fragment identifiers or `303 See Other` redirects to associate a resource URI to a document *about* it (as opposed to a document it *identifies* or *addresses*). Furthermore, recipes have been proposed for handling content negotiation such that when dereferencing a URI, clients can request RDF in a suitable format using HTTP `Accept` headers. Finally, the guidelines recommend providing as full an RDF description as possible about a resource upon dereferencing, particularly locally available triples where that resource is mentioned in the subject or object position.

Linking Aliases: On the Web, it is common practice for multiple publishers to speak about the same entities in different locations. One option to support the interoperability of data overlapping in this manner would be to use consistent URIs to refer to the same thing, allowing contributions from the different parties to be merged about the entities in question. However, if there were only one URI for, e.g., `Citrus` on the Web, that URI could only dereference to one document in one location. Hence, Linked Data guidelines (and the Semantic Web standards) allow for the use of multiple URI aliases that refer to the same thing. Subsequently, Linked Data guidelines recommend using `owl:sameAs` links to specify that a remote location speaks about the same resource using an alternative identifier (see Example 9 for a real-life example of aliases linked across two Linked Data publishers: DBpedia and Freebase).

Describing Vocabularies Terms: In Section 1.4, we gave a summary of the semantics of RDF and how standards such as RDFS and OWL can be used to unambiguously define the meaning of terms, particularly classes and properties. Linked Data guidelines recommend the shared use of common vocabularies of class and property terms (including, e.g., FOAF for information about people). The semantics of these vocabularies can be described in RDF using the RDFS and OWL languages, and can be mapped to related vocabularies, where these descriptions can be dereferenced by looking up the URI of the term (see Example 8 on how the semantics of RDFS terms enables automated inference).

Provision of SPARQL Endpoints: Linked Data guidelines do not require publishers to provide SPARQL endpoints over their content: dereferenceable documents are sufficient to constitute Linked Data. However, the provision of a SPARQL endpoint for a given Linked Data site gives consumers a single-point-of-access to the merge of contributions on that site. Furthermore, SPARQL engines are often used to dynamically generate dereferenced documents for consumers. Hence, public SPARQL endpoints are often provided by Linked Data publishers alongside dereferenceable RDF documents.

The Linking Open Data community has been successful in engaging with publishers of Open Data and encouraging them to follow Linked Data principles and related best-practices. Publishers of Linked Data now include such household names as the BBC, the New York Times, Sears, Freebase (owned by Google), the UK government, and so forth, making data described using Semantic Web standards openly available on the Web.

To keep track of the growth in published datasets made available as Linked Data, in 2007, Cyganiak and Jentzsch first began collating the "Linking Open Data cloud diagram" (or "LOD cloud" for short). The most recent version of the diagram is depicted in Figure 1.2: each node represents a composite dataset, with directed edges indicating links between datasets. For a dataset to be eligible for the diagram, it must use dereferenceable URIs that resolve to RDF documents, it must contain at least one thousand triples, and it must contain at least fifty links to an external dataset. The cloud depicted contains a total of 295 such datasets. The size of each node indicates the number of triples that the dataset contains, varying from less than ten thousand RDF triples for the smallest nodes to over one billion triples for the largest nodes. At the center of the diagram is DBpedia, which "mints" a dereferenceable URI for every entity described by its own Wikipedia article and offers an RDF description thereof, publishing an aggregate total of 1.89 billion triples about 20.8 million entities (extracted from multi-lingual versions of Wikipedia).

The colors of the diagram indicate different high-level domains under which the datasets fall: media, geographic, publications, user-generated content, government, cross-domain and life sciences. These are domains within which there has traditionally been an emphasis on publishing Open Data: for example, governmental organizations are interested in Open Data to increase transparency and allow developers to combine and build services over public-domain knowledge (e.g., census data, traffic levels, pollution, etc.) for the benefit of citizens; within the life sciences, the field of bioinformatics is struggling with colossal amounts of raw data on genomes, diseases, drugs, etc., coming from a variety of sources; and so forth. Aside from these specialized domains, there is also a wealth of more general-interest datasets in the cross-domain, media and user-generated content categorizations.

And so we now we have the first real foundations for a true Web of Data, composed of a wide selection of datasets described using a common formal structure (RDF), defined using extensible vocabularies founded on formal languages with well-defined semantics (RDFS and OWL), all published openly on the Web and interlinked to allow for automated discovery and navigation of related information, with many datasets indexed in engines that provide a standard, flexible query functionality (SPARQL).[28] Although some may argue that this Web of Data is still a long way off the intangible Semantic Web promised in years gone by (as described in Section 1.2 and epitomized by the

[28] A total of 201 datasets (68%) claim to offer access to a SPARQL endpoint: http://wifo5-03.informatik.uni-mannheim.de/lodcloud/state/; retr. 2012/12/02.

layer cake), it inarguably represents the first tangible deployment of Semantic Web standards on the Web itself.

But the war is far from won: although the Web infrastructure has proven successful in being able to host this massive new publishing initiative, the challenges faced when *consuming* this Web of Data—when harnessing its content for the purposes of building applications—are only now becoming clear. In particular, querying this novel Web of Data requires new techniques and new ways of thinking forged upon the expertise collected in related areas such as databases, distributed computing and information retrieval. The rest of this book focuses on these core challenges—challenges that must be addressed before the true potential of this fledgling Web of Data can (finally) be unlocked.

Chapter Acknowledgements:

The author of this chapter was funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2). I'd also like to thank the anonymous reviewer whose comments helped improve this chapter.

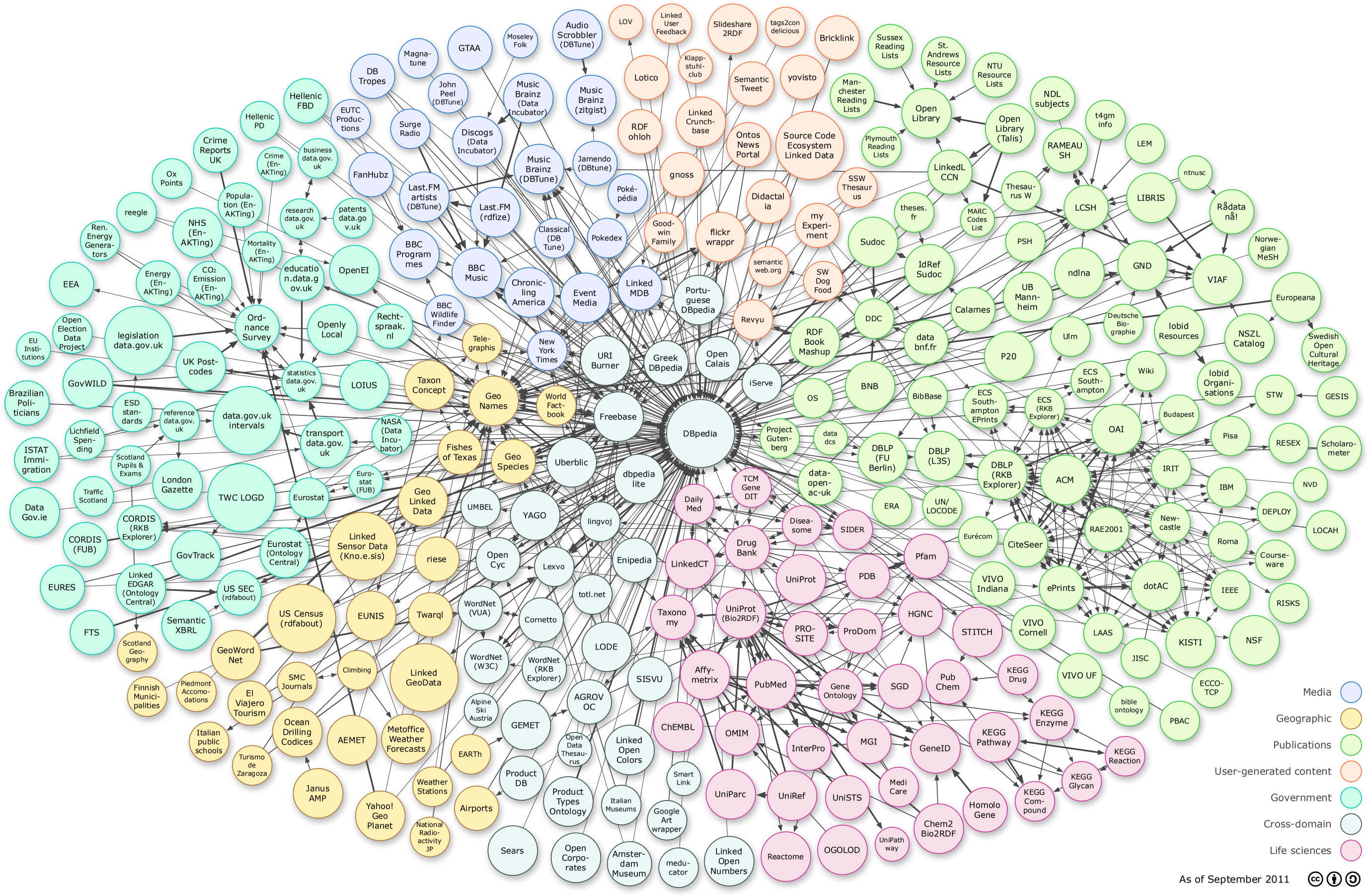


FIGURE 1.2: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

Bibliography

[1] Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman. RDFa Core 1.1. W3C Recommendation, June 2012. http://www.w3.org/TR/rdfa-syntax/.

[2] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation, October 2008. http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application.* Cambridge University Press, 2002.

[4] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-syntax-grammar/.

[5] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Turtle: Terse RDF Triple Language. W3C Working Draft, July 2012. http://www.w3.org/TR/turtle/.

[6] Tim Berners-Lee. Semantic Web Road map. W3C Design Issues, September 1998. http://www.w3.org/DesignIssues/Semantic.html.

[7] Tim Berners-Lee. Linked Data. W3C Design Issues, July 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[8] Tim Berners-Lee and Dan Connolly. Notation3 (N3): A readable RDF syntax. W3C Team Submission, March 2011. http://www.w3.org/TeamSubmission/n3/.

[9] Tim Berners-Lee, Jim Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 5(284):35–40, 2001.

[10] Diego Berrueta and Jon Phipps. Best Practice Recipes for Publishing RDF Vocabularies. W3C Working Group Note, August 2008. http://www.w3.org/TR/swbp-vocab-pub/.

[11] Mark Birbeck and Shane McCarron. CURIE Syntax 1.0: A syntax for expressing Compact URIs. W3C Working Group Note, December 2010. http://www.w3.org/TR/curie/.

[12] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – a crystallization point for the Web of Data. *J. Web Sem.*, 7(3):154–165, 2009.

[13] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-schema/`.

[14] Dan Brickley, R.V. Guha, and Andrew Layman. Resource Description Framework (RDF) Schemas. W3C Recommendation, April 1998. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[15] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. SPARQL Protocol for RDF. W3C Recommendation, January 2008. `http://www.w3.org/TR/rdf-sparql-protocol/`.

[16] Li Ding and Tim Finin. Characterizing the Semantic Web on the Web. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 2006.

[17] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[18] Paul Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 Update. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-update/`.

[19] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. OWL: Yet to arrive on the Web of Data? In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *LDOW*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[20] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Candidate Recommendation, March 2013. `http://www.w3.org/TR/sparql11-entailment/`.

[21] Birte Glimm and Sebastian Rudolph. Status QIO: Conjunctive query entailment is decidable. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *KR*. AAAI Press, 2010.

[22] Christine Golbreich and Evan K. Wallace. OWL 2 Web Ontology Language: New Features and Rationale. W3C Recommendation, October 2009. `http://www.w3.org/TR/owl2-new-features/`.

[23] Jan Grant and Dave Beckett. RDF Test Cases. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-testcases/`.

[24] Bernardo Cuenca Grau, Boris Motik, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language: Profiles. W3C Recommendation, October 2009. `http://www.w3.org/TR/owl2-profiles/`.

[25] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.

[26] Harry Halpin, Patrick J. Hayes, James P. McCusker, Deborah L. McGuinness, and Henry S. Thompson. When owl:sameAs isn't the same: An analysis of identity in Linked Data. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2010.

[27] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-query/`.

[28] Jonathan Hayes and Claudio Gutiérrez. Bipartite graphs as intermediate model for RDF. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2004.

[29] Patrick Hayes. RDF Semantics. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-mt/`.

[30] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[31] James Hendler and Deborah L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.

[32] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004. `http://www.w3.org/Submission/SWRL/`.

[33] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the Semantic Web. In *AAAI/IAAI*, pages 792–797, 2002.

[34] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation, December 2004. `http://www.w3.org/TR/webarch/`.

[35] Ilianna Kollia, Birte Glimm, and Ian Horrocks. SPARQL query answering over OWL ontologies. In Grigoris Antoniou, Marko Grobelnik, Elena

Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *ESWC (1)*, volume 6643 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2011.

[36] Markus Krötzsch, Frederick Maier, Adila Krisnadhi, and Pascal Hitzler. A better uncle for OWL: nominal schemas for integrating rules and ontologies. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *WWW*, pages 645–654. ACM, 2011.

[37] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Description Logic Rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 80–84. IOS Press, 2008.

[38] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A Description Logic Primer. *CoRR*, abs/1201.4089, 2012.

[39] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, February 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[40] Sean Luke, Lee Spector, David Rager, and James A. Hendler. Ontology-based Web Agents. In *Agents*, pages 59–66, 1997.

[41] Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres. On blank nodes. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 421–437. Springer, 2011.

[42] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-primer/`.

[43] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-features/`.

[44] Alistair Miles, Thomas Baker, and Ralph Swick. Best Practice Recipes for Publishing RDF Vocabularies. W3C Working Draft, March 2006. `http://www.w3.org/TR/2006/WD-swbp-vocab-pub-20060314/` (Later superseded by [10]).

[45] Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. Simple and efficient minimal RDFS. *J. Web Sem.*, 7(3):220–234, 2009.

[46] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[47] Axel Polleres. From SPARQL to rules (and back). In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 787–796. ACM, 2007.

[48] Axel Polleres. How (well) do Datalog, SPARQL and RIF interplay? In Pablo Barceló and Reinhard Pichler, editors, *Datalog*, volume 7494 of *Lecture Notes in Computer Science*, pages 27–30. Springer, 2012.

[49] Eric Prud'hommeaux and Carlos Buil-Aranda. SPARQL 1.1 Federated Query. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-federated-query/`.

[50] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[51] Sebastian Rudolph. Foundations of Description Logics. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F. Patel-Schneider, editors, *Reasoning Web*, volume 6848 of *Lecture Notes in Computer Science*, pages 76–136. Springer, 2011.

[52] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[53] Michael Schneider. OWL 2 Web Ontology Language RDF-Based Semantics. W3C Recommendation, October 2009. `http://www.w3.org/TR/owl2-rdf-based-semantics/`.

[54] Andy Seaborne. SPARQL 1.1 Query Results CSV and TSV Formats. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-results-csv-tsv/`.

[55] Andy Seaborne. SPARQL 1.1 Query Results JSON Format. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-results-json/`.

[56] Manu Sporny. JSON-LD Syntax 1.0. W3C Last Call Working Draft, April 2013. `http://www.w3.org/TR/json-ld-syntax/`.

[57] Patrick Stickler. CBD – Concise Bounded Description. W3C Recommendation, June 2005. `http://www.w3.org/Submission/CBD/`.

[58] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *J. Web Sem.*, 3(2–3):79–115, 2005.

[59] Denny Vrandečíc, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Leveraging non-lexical knowledge for the Linked Open Data Web. *Review of April Fool's day Transactions (RAFT)*, 5:18–27, 2010.

[60] Gregory Todd Williams. SPARQL 1.1 Service Description. W3C Recommendation, March 2013. `http://www.w3.org/TR/sparql11-service-description/`.

[61] David Wood, Stefan Decker, and Ivan Herman, editors. *Proceedings of the W3C Workshop – RDF Next Steps, Stanford, Palo Alto, CA, USA, June 26–27*. Online at `http://www.w3.org/2009/12/rdf-ws/`, 2010.