12 Ontologies and Knowledge Representation

Knowledge representation (KR) systems are an important component of many artificial intelligence applications, such as planners, robots, natural language processors, and game-playing systems. The KR system stores the underlying model of the domain that is employed by the application. In contrast to database systems, KR systems need to represent more complex and possibly incomplete domain models. KR systems use reasoning techniques to answer queries about the knowledge. Knowledge representation is the branch of artificial intelligence that studies the design and expressive power of KR languages, the computational properties of their associated reasoning problems, and effective implementation of KR systems.

Some aspects of data integration can also be viewed as a knowledge representation problem. Data sources and their contents lend themselves to rather complex modeling. Determing the relationships between data sources, or between a data source and a mediated schema, often requires subtle reasoning. For these reasons researchers have considered applying knowledge representation techniques to data integration from the very early days of the field. This chapter describes some of the principles of knowledge representation and how they apply to data integration.

Section 12.1 motivates the use of knowledge representation in data integration with an example. In Section 12.2 we describe description logics, which is the main family of KR languages that has been employed in the context of data integration. Description logics offer a set of logical formalisms for defining a domain *ontology*: a description of the entities in the domain, relationships between them, and any other known constraints. The key question explored in description logics is the trade-off between expressive power and the complexity of performing inference.

In recent years, much of the work in knowledge representation has been conducted under the umbrella of the *Semantic Web*. The Semantic Web is a vision that calls for associating semantic markup with content on the Web. The markup can be used to enhance the accuracy of search and to enable novel services that combine information from multiple sources on the Web. Section 12.3 describes the Semantic Web vision and gives an overview of some of the standards that have been developed to enable the vision, such as the Resource Description Language (RDF) and the Web Ontology Language (OWL), which is based on description logics.

Principles of Data Integration, DOI: 10.1016/B978-0-12-416044-6.00012-0

12.1 Example: Using KR in Data Integration

We begin with an example that illustrates the power of knowledge representation and reasoning in data integration. In the example we use a few technical terms informally, but we explain them in later sections.

Consider an ontology of movies, shown in Figure 12.1. The ontology has the class Movie representing the set of all movies. The class Movie has two subclasses: Comedy and Documentary, and they are declared to be disjoint from each other. Note that ontologies permit overlapping classes, whereas object-oriented database models typically do not. The ontology also includes a set of properties, representing relationships between pairs of objects in the domain. In our example, we have the properties award and oscar, and we declare that oscar is a subproperty of award.

The ontology will serve as the mediated schema of the data integration system. Hence, we describe data sources in terms of the classes and properties in the ontology and pose queries using these terms as well. Let us consider several queries and data sources and see how reasoning serves as a mechanism for determining when a data source is relevant to a given query.

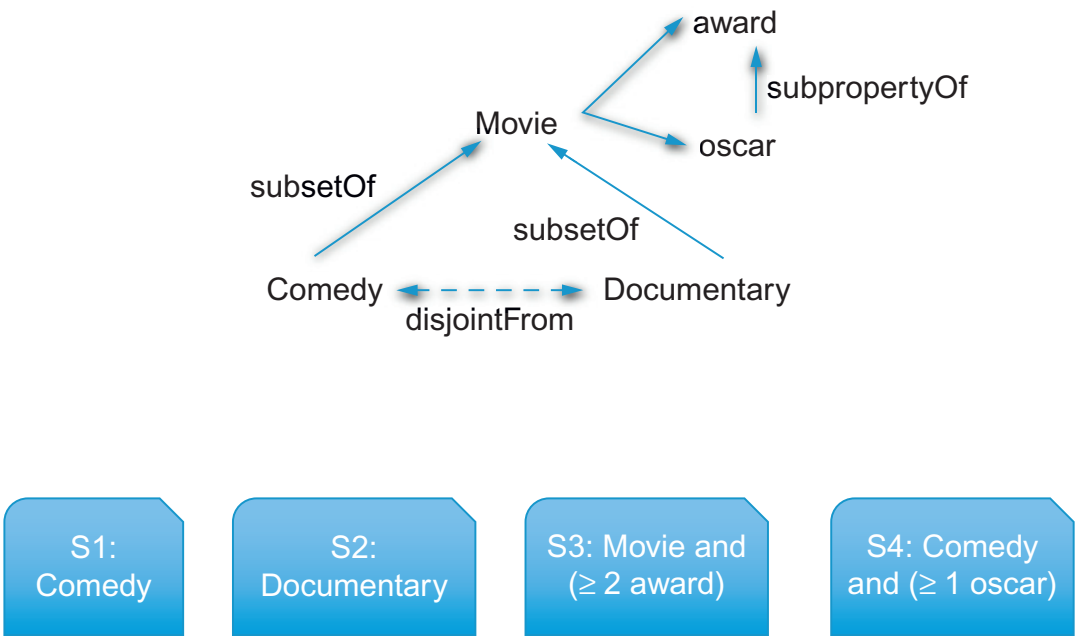

FIGURE 12.1 The figure shows a simple ontology of movies, with subclasses for comedies and documentaries. Note that the concepts **Comedy** and **Documentary** are stated to be disjoint. The property **oscar** is a subproperty of **award**. Given this ontology, the system can reason about the relevance of individual data sources to a query. For example, **S1** is relevant to a query asking for all movies, but **S2** is not relevant to a query asking for comedies. Sources **S3** and **S4** are relevant to queries asking for all movies that won awards, and **S4** is relevant to a query asking for movies that won an Oscar.

In the simplest example, consider the data source S_1 that provides comedies, as specified by the following LAV description, and the query Q_1 asking for all movies:

$S_1(\mathsf{X}) \subseteq \mathsf{Comedy}(\mathsf{X})$
$Q_1(\mathsf{X})$:- $\mathsf{Movie}(\mathsf{X})$

Since Comedy is a subclass of Movie, simple reasoning will deduce that data source S_1 is relevant to Q_1. In a similar fashion, we can infer that certain sources can be pruned from consideration. For example, suppose we have a data source S_2 containing documentaries and a query Q_2 asking for comedies. We can deduce that S_2 is irrelevant to Q_2.

$S_2(\mathsf{X}) \subseteq \mathsf{Documentary}(\mathsf{X})$
$Q_2(\mathsf{X})$:- $\mathsf{Comedy}(\mathsf{X})$

The two examples above only involved reasoning about basic classes and the relationships between them. We can go further and reason about *complex class expressions*. Suppose we have the data source S_3 that contains movies that won at least two awards, and a query, Q_3, asking for all movies that won an award:

$S_3(\mathsf{X}) \subseteq \mathsf{Movie}(\mathsf{X}) \sqcap (\geq 2\ \mathsf{award})(\mathsf{X})$
$Q_3(\mathsf{X})$:- $\mathsf{Comedy}(\mathsf{X}), \mathsf{award}(\mathsf{X},\mathsf{Y})$

The reasoning system can translate the query into the expression $\mathsf{Movie} \sqcap (\geq 1\ \mathsf{award})$, i.e., movies that won at least *one* award. Therefore, since the class of movies that won two awards is a subset of the class of movies that won at least one award, the system can deduce that source S_3 is relevant to the query Q_3. Note that numerical restrictions can also be used to derive disjointness between classes. For example, suppose we had a data source that supplied only movies without awards, described by the complex class expression $\mathsf{Movie} \sqcap (\leq 0\ \mathsf{award})$. We would be able to prune this source from consideration for Q_3.

Finally, the system can also use relationships between properties in its reasoning. For example, suppose we have a data source, S_4, that only contains Oscar-winning comedies.

$S_4(\mathsf{X}) \subseteq \mathsf{Comedy}(\mathsf{X}) \sqcap (\geq 1\ \mathsf{oscar})(\mathsf{X})$

We could still deduce that S_4 is relevant to Q_3 because Comedy is a subclass of Movie and oscar is a subproperty of award.

As we can see from the above examples, the ontology can represent fairly complex relationships between data sources, the mediated schema, and queries. The inference engine can reason about these relationships to decide whether data sources are relevant to given queries. In the next section we describe more formally how these services are achieved.

12.2 Description Logics

Description logics are a family of languages for defining ontologies that vary in their expressive power. They range from languages with limited expressive power for which reasoning is very efficient, to very expressive languages for which reasoning is intractable

or even undecidable. Formally, a description logic is a subset of first-order logic that is restricted to unary relations (called concepts or classes) and binary relations (called roles or properties). The concepts describe sets of individuals in the domain, and the roles describe binary relationships between individuals.

A description logic knowledge base contains two parts: a *Tbox*, which is a set of statements defining concepts and roles, and an *Abox*, which is a set of assertions on individuals in the domain. The Tbox defines two kinds of concepts, *base* concepts and *complex* concepts, that are defined from the base concepts using a set of *constructors*. For example, the concept $\mathsf{Comedy} \sqcap (\geq 1\ \mathsf{award})$ denotes the individuals that are comedies and won at least one award and is defined using the constructors $\sqcap$ (denoting conjunction) and $(\geq \mathsf{n}\ \mathsf{R})$ (denoting number restrictions).

Description logics differ from each other on the set of constructors they allow to define complex concepts and roles. In principle, every choice of constructors leads to a different language, though some combinations of constructors can lead to languages that are equivalent in expressive power. The set of constructors allowed by a description logic has significant impact on the complexity of the reasoning tasks offered by the logic.

In what follows we describe a relatively simple description logic, $\mathcal{L}$, for which reasoning is tractable. We begin with describing the syntax of $\mathcal{L}$.

12.2.1 Syntax of Description Logics

A knowledge base, Δ, in $\mathcal{L}$ is composed of a Tbox, $\Delta_{\mathcal{T}}$ and an Abox $\Delta_{\mathcal{A}}$. Assertions in $\Delta_{\mathcal{T}}$ have two possible forms: definitional assertions of the form $\mathsf{A} := \mathsf{C}$ or inclusion assertions of the form $\mathsf{A} \sqsubseteq \mathsf{C}$. In both cases, we require that the left-hand side, A, be a base concept, while C can be a complex concept. More expressive description logics allow both sides of an assertion to be complex concepts.

Complex concepts are defined using the following grammar, where A denotes a base concept and C, D denote complex concepts:

$C, D \rightarrow$	$A \mid$	(base concept)
	$\top \mid$	(top, the set of all individuals)
	$C \sqcap D \mid$	(conjunction)
	$\neg A \mid$	(complement)
	$\forall R.C \mid$	(universal quantification)
	$(\geq nR) \mid (\leq nR)$	(number restrictions)

Example 12.1

Consider the following ontology, δ, of movies. The following are assertions in its Tbox, $\delta_{\mathcal{T}}$:

a_1. Italian $\sqsubseteq$ Person
a_2. Comedy $\sqsubseteq$ Movie
a_3. Comedy $\sqsubseteq \neg$Documentary

a_4. Movie $\sqsubseteq$ ($\leq$ 1 director)
a_5. AwardMovies := Movie $\sqcap$ ($\geq$ 1 award)
a_6. ItalianHits := AwardMovie $\sqcap$ ($\forall$director.Italian)

The first assertion defines Italians to be a subset of people. Assertions a_2 and a_3 specify that comedies are movies and they are disjoint from documentaries. The assertion a_4 specifies that movies have at most one director. The concept AwardMovies is defined, by a_5, to be the set of movies that won at least one award, and a_6 defines the concept ItalianHits to be all the movies that won at least one award for which all directors are Italian.

Assertions in the Abox, $\Delta_{\mathcal{A}}$, are statements about individuals in the domain, much like tuples in a database with arity of 1 or 2. There are two kinds of assertions. The first kind, of the form C(a), asserts that the individual a is an instance of the concept C. The second kind of assertion, of the form R(a,b), asserts that the role R holds between the objects a and b. The constant b is called a *filler* of the role R for a.

An important difference between Aboxes and databases is that an assertion of the form C(a) can involve a complex concept C. In effect, asserting C(a) is akin to stating that an individual a is in the extension of a view. As we discuss later, we can draw several conclusions from such assertions.

Example 12.2

Consider an Abox, $\delta_{\mathcal{A}}$, for the Tbox defined in Example 12.1:

Comedy(LifeIsBeautiful)
director(LifeIsBeautiful, Benigni)
Italian(Benigni)
award(LifeIsBeautiful, Oscar1997)
ItalianHits(LaStrada)

The reasoning engine would be able to deduce from the first four assertions that the movie LifeIsBeautiful is an instance of the concept ItalianHits. The last assertion only tells us that LaStrada is a member of the concept ItalianHits, but does not tell us who the director is or what awards the movie received. However, we do know that the director is Italian and it won at least one award.

12.2.2 Semantics of Description Logics

The semantics of description logic knowledge bases are given via *interpretations*. An interpretation, I, contains a nonempty domain of objects, $\mathcal{O}^I$. The interpretation assigns an object a^I to every individual a mentioned in $\Delta_{\mathcal{A}}$. Description logics make the *unique-names assumption*: for any distinct pairs of individuals, a and b, $a^I \neq b^I$. (Note that databases also make the unique-names assumption.)

The interpretation I assigns a unary relation, C^I, to every concept in $\Delta_\mathcal{T}$ and a binary relation, R^I, over $\mathcal{O}^I \times \mathcal{O}^I$ to every role R in $\Delta_\mathcal{T}$. The extensions of concept and role descriptions are given by the following equations ($\sharp\{S\}$ denotes the cardinality of a set S):

$$\top^I = \mathcal{O}^I,$$
$$(C \sqcap D)^I = C^I \cap D^I,$$
$$(\neg A)^I = \mathcal{O}^I \setminus A^I,$$
$$(\forall R.C)^I = \{d \in \mathcal{O}^I \mid \forall e : (d,e) \in R^I \rightarrow e \in C^I\}$$
$$(\geq\ nR)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \geq n\}$$
$$(\leq\ nR)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \leq n\}$$

For example, the first two lines define the extension of $\top$ to be the set of all objects in $\mathcal{O}^I$, and the extension of $(C \sqcap D)$ to be the intersection of the extensions of C and D. The extension of $(\forall R.C)$ is the set of objects in the domain for which all the fillers on the role R are members of C. The extension of $(\geq\ nR)$ is the set of objects in the domain that have at least n fillers on the role R.

The assertions in a knowledge base specify a set of constraints on the interpretations. The interpretations that satisfy these constraints are called *models*. Intuitively, models describe states of the domain that are possible given the assertions in the knowledge base.

Definition 12.1 (Containment and Equivalence). *An interpretation, I, for a knowledge base Δ is a model of Δ if*

- $A^I \subseteq C^I$ *for every inclusion* $A \sqsubseteq C$ *in* $\Delta_\mathcal{T}$,
- $A^I = C^I$ *for every statement* $A := C$ *in* $\Delta_\mathcal{T}$,
- $a^I \in C^I$ *for every statement* $C(a)$ *in* $\Delta_\mathcal{A}$, *and*
- $(a^I, b^I) \in R^I$ *for every statement* $R(a,b)$ *in* $\Delta_\mathcal{A}$. □

Example 12.3

Consider the following interpretation for our example knowledge base δ. For simplicity, assume that I is the identity mapping on the constants in δ. Let $\mathcal{O}$ be the set

{LifeIsBeautiful, LaStrada, Benigni, Director1, Oscar1997, Award1, Actor1, Actor2}.

Note that Director1, Award1, Actor1, and Actor2 are not mentioned in the Abox of δ but are necessary in $\mathcal{O}$ in order to obtain a model. Consider the following extensions for the concepts in δ_T (we omit the parentheses around unary tuples):

Movie^I:	{LifeIsBeautiful, LaStrada}
Comedy^I:	{LifeIsBeautiful}
Documentary^I:	∅
Person^I:	{Benigni, Director1}
Italian^I:	{Benigni, Director1}
award^I:	{(LifeIsBeautiful, Oscar1997), (LaStrada, Award1)}
director^I:	{(LifeIsBeautiful, Benigni), (LaStrada, Director1)}
actor^I:	{(LaStrada, Actor1), (LaStrada, Actor2)}

The reader can verify that all the assertions in the Tbox and Abox are satisfied, and therefore I is a model of δ. Two points are worth noting. First, we do not know who the director of LaStrada is or the award it received, so the interpretation I only includes the constants that are needed to satisfy the Tbox. Also, note that LifeIsBeautiful is a member of Comedy, which is not asserted in the knowledge base but is consistent with it. If we remove LifeIsBeautiful from the extension of Comedy, we will still have a model of δ.

Small changes to I would preclude it from being a model of δ. For example, if we add LifeIsBeautiful to the extension of the concept Documentary then a_3 would no longer be satisfied, and if we add another director to either of the movies, then a_4 would no longer be satisfied.

12.2.3 Inference in Description Logics

A description logic system supports two main kinds of inference: subsumption and query answering.

Subsumption: Given two concepts, C and D, the fundamental question is whether C is subsumed by D. Intuitively, C is subsumed by D if in every model of the knowledge base, the instances of C are always guaranteed to be instances of D. As we discuss later, subsumption is much like query containment (see Chapter 2), but the reasoning patterns are somewhat different. Formally, subsumption is defined as follows.

Definition 12.2 (Subsumption). *We say that the concept* C *is subsumed by the concept* D *w.r.t. a Tbox* $\Delta_{\mathcal{T}}$ *if* $C^I \subseteq D^I$ *in every model* I *of* $\Delta_{\mathcal{T}}$. □

Example 12.4

The concept Movie $\sqcap$ ($\geq$ 2 award) is subsumed by AwardMovies in δ, but it is not subsumed by ItalianHits. The concept Movie $\sqcap$ ($\forall$director.Person) subsumes the concept ItalianHits, because the restriction on the director is weaker, and there is no restriction on the number of awards.

As we saw earlier, subsumption can be used to infer relationships between pairs of data sources, between a concept in the mediated schema and a data source, or between a query and a data source. For example, if we pose a query on the data integration system to find all Italian movies, and we have a source S providing instances of the concept ItalianHits, then we can infer that S is relevant to the query. On the other hand, if the query asks for Italian movies that received at least two awards, then we can only use S if it also provides the number of awards won by each movie, so we can filter those that have at least two awards.

In the logic $\mathcal{L}$ that we described above, subsumption can be done in time that is polynomial in the size of the Tbox. However, slight additions to the logic would cause subsumption to be intractable or even undecidable. As a simple example, if we allow

the disjunction constructor that defines a complex concept to be the union of two other concepts, then reasoning becomes NP-complete. Adding negation on complex concepts would have the same effect. Some of the other constructors that have been considered are existential quantification: (∃R C) specifying that at least one of the fillers of R needs to be a member of C; and sameAs specifying that the fillers on role-chain $P_1,\ldots,P_n$ need to be the same as the fillers on the role-chain $R_1,\ldots,R_n$. In fact, if the sameAs constructor is allowed on nonfunctional roles, then subsumption becomes undecidable.

Query answering: The second class of inference problems considers the Tbox and Abox. The most basic reasoning task is *instance checking*. We say that $C(a)$ is entailed by a knowledge base Δ, denoted by $\Delta \models C(a)$, if in every model I of Δ, $a^I \in C^I$.

When description logics are used for data integration, more attention has been paid to the more general inference problem of answering conjunctive queries over knowledge bases. Formally, the problem is defined as follows.

Definition 12.3 (conjunctive queries over Description Logic knowledge bases). *Let Δ be a knowledge base, and let Q be a conjunctive query of the form*

$$q(\bar{X}) :- p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)$$

where $p_1,\ldots,p_n$ are either concepts or roles in $\Delta_{\mathcal{T}}$.

The set of answers to Q over Δ is defined as follows. Given an interpretation I of Δ we can evaluate Q over I as if it were a database. We denote by $Q(I)$ the set of tuples obtained by evaluating Q over I. A tuple $\bar{t}$ of constants in $\Delta_{\mathcal{A}}$ is in the answer of Q over Δ if for every model, I, of Δ, $\bar{t} \in Q(I)$. □

The reader should observe the similarity between Definition 12.3 and the definition of certain answers in Chapter 3.2.1.

Example 12.5

Consider a simple example where we have the query

$Q1(X)$:- Comedy(X), ItalianHits(X), award(X, Y)

and suppose we have the Abox with the following facts:

ItalianHits(LifeIsBeautiful), Comedy(LifeIsBeautiful)

If we simply apply the query to the Abox, there will be no answers, because we do not have facts for the award relation. However, ItalianHits implies that the movie has at least one award, and therefore that the subgoal award(X, Y) will be satisfied in every model of the knowledge base. Hence, LifeIsBeautiful should be in the answer to $Q1$.

The example above can be handled by removing subgoals of the query that can be deemed redundant based on the ontology (in this example, award(X, Y) is entailed by ItalianHits(X) and therefore is redundant). However, consider the following query that is a union of two conjunctive queries.

$Q2(X)$:- Movie(X), ≥ 1 award(X)
$Q2(X)$:- Comedy(X), ∀director.Italian(X), ≤ 0 award(X)

Now suppose we have the following Abox:

Comedy(LaFunivia), director(LaFunivia, Antonioni), Italian(Antonioni)

The first conjunctive query will not yield any answers, even if we realize that a comedy is also a movie, because we know nothing about the awards that LaFunivia won. The second conjunctive query will also yield no answers but for a different reason: just because the Abox does not contain any awards for LaFunivia does not mean that the movie did not win any awards! However, by reasoning about *both* queries together, we can infer that the following conjunctive query can be added to *Q*2 without losing correctness.

*Q*2(X) :- Comedy(X), ∀director.Italian(X)

The correctness of the last conjunctive query is established by reasoning by cases. If a movie is an Italian comedy, it has either no awards or at least one award. In the first case, it would satisfy the second conjunctive query, and in the second case it would satisfy the first conjunctive query. Hence, it does not matter how many awards an Italian comedy has.

Applying the new conjunctive query to the Abox would still not yield LaFunivia as an answer. First, the system would need to infer that (∀director.Italian(LaFunivia) holds, from the fact that the movie has exactly one director and he is known to be Italian.

Finding all the answers for a conjunctive query over a description logic knowledge base can be rather tricky. While it can still be done in polynomial time in the size of the Abox for fairly expressive description logics, it requires more complex algorithms. Answering recursive datalog queries is even more complicated. In fact, even for $\mathcal{L}$, answering recursive queries is undecidable.

12.2.4 Comparing Description Logics to Database Reasoning

To conclude this section, it is worthwhile to consider more carefully the relationship between description logics and view mechanisms in database systems. Complex concepts are similar to views in the sense that they are defined by a set of constructors from base relations. In the case of relational views, the constructors are join, selection, projection, union, and some forms of negation. The analog of subsumption reasoning is query containment, which we covered in Chapter 2.3. There are, however, significant differences between the two formalisms.

The first difference is that the set of constructors used in description logics enable defining views that are impossible or complicated to define in datalog or SQL. For example, description logics enable stating that one concept is the complement of another, which is not possible with database views. One can define a view to be the complement of another by using negation, but then query containment becomes quickly undecidable. Similarly, universal restrictions, such as (∀ director.Italian), also require negation and composition in a database view language. Finally, number restrictions (≥ 3 actor) and (≤ 2 actor) are difficult to specify with views. We can specify minimal cardinality restrictions, (≥ n R), with a conjunctive query using the ≠ predicate, but the length of the expression is dependent

on n. Specifying maximal cardinality restrictions, $(\leq n\ R)$, requires negation and a similarly long expression.

On the other hand, conjunctive queries enable defining views with arbitrary joins and are not limited to unary and binary relations. In description logics the only join relationships possible are constructed by following role paths. Hence, the main lesson from description logics is that by restricting attention to unary and binary predicates, and by limiting the joins one can perform, description logics offer a view definition language for which containment is decidable (and often efficient) for a class of views for which it would not be possible otherwise.

The second difference is that in a description logic Abox we assert facts about *any* concept, whether it is a base concept or complex one. In the latter case, asserting a fact is akin to stating that a tuple is in the instance of a view. The description logic is able to make inferences about such facts as well. For example, consider the concept ItalianHits in our running example. In δ we asserted that LaStrada is an instance of ItalianHits. The system can infer that if LaStrada has a director, then that director is an instance of the concept Italian even though we do not know who the director is. Similarly, we can deduce that the movie won at least one award, even if we do not know which. Relatedly, unlike database systems, description logic knowledge bases make the *open-world assumption*. If a fact is not explicitly in the knowledge base, that does not mean that the fact does not hold.

The analog to reasoning about instances of complex concepts in database formalisms would be techniques for answering queries using views, which we covered in Chapter 2.4. For example, we could have a view for Italian hits that includes the instance LaStrada. Techniques for answering queries using views would be able to conclude that LaStrada would be in the answer to the query asking for all movies that won at least one award. As before, description logics offer a setting in which we can use a rich set of constructors and still be able to develop algorithms for answering queries using views.

The final difference we mention is that description logics unify the schema definition language with the constraint language. For example, some description logics allow arbitrary inclusion statements of the form $C \sqsubseteq D$, where both C and D are complex concepts. The reasoning algorithms would consider these constraints as well in making inferences, whereas query containment under constraints is fairly limited.

It is also worthwhile to compare object-oriented database schemas and description logics. While both formalisms model domains in terms of classes of objects and their properties, there are fundamental differences between them. Object-oriented schemas focus on the physical properties of objects. They define which properties are attached to an instance of a class. The only form of reasoning is inheritance of properties from classes to their subclasses. Furthermore, in most OODB formalisms an object can be a member of at most one class (and its superclasses). Description logics focus on providing a language for expressing relationships between classes and making inferences based on partial information. As a result, objects can belong to multiple classes. Description logic knowledge bases specify very little about the physical properties of the objects or how they are to be stored.

12.3 The Semantic Web

The Semantic Web is a vision in which documents on the Web have annotations that make their meanings explicit. Today, the Web is largely text and links between HTML *pages*. However, the entities and the relationships that are the subjects of the pages are not clearly described. The lack of such annotations hinders the accuracy of search and the ability to create more advanced services. The following example illustrates the potential of such annotations.

Example 12.6

Consider pages concerning movies and their reviews on the Web. For the most part, a page would come up in an answer to a query if the words on the page match the words in the query. Furthermore, the proximity between the occurrences of the query words on the page would also be an important feature in ranking the page. Consider a page that has multiple movie reviews, ordered alphabetically by the title of the movie, but the word "review" only appears once near the top of the page. The page is unlikely to come up in the answer to the query "review zanzibar," because the words "review" and "zanzibar" occur far away from each other on the page, and the search engine doesn't *know* that the page contains review objects.

Suppose that in addition to the text on the page, we would also add an annotation to each review specifying that it is a movie review, along with the title of the movie, the name of the reviewer, and possibly even the sentiment of the review. We could now add functionality to search engines to exploit these annotations for more accurate retrieval of reviews. If the practice of annotation were prevalent, we could even support queries such as "find all reviews of zanzibar," which would retrieve the appropriate segments of *multiple* pages on the Web. In addition, we can relate the review information to other data relating to the movie, such as playing times, trailers, and information about the actors.

The vision of the Semantic Web is clearly a very ambitious one, and that ambition has led to quite a bit of criticism. There are two main challenges faced by the Semantic Web. The first challenge is to entice people to invest the extra effort to create these annotations without an immediate reward. The second challenge concerns the semantic heterogeneity issues that will arise when so many people create structured representations of data. While it seems like a long shot that the vision, in its entirety, will be fulfilled, there have been some developments that facilitate structural markup on the Web. These developments have had an impact in certain domains and within enterprises. At the core of these developments is the Resource Description Framework (RDF) for semantic markup and its associated schema languages, RDF Schema (RDFS) and the Web Ontology Language (OWL). These standards are built on the principles of knowledge representation languages and adapt them to the context of the Web, where knowledge is authored in a decentralized fashion by many participants. We describe the key ideas from these standards in the next few sections.

12.3.1 Resource Description Framework

RDF is a language for describing resources on the Web. Initially, RDF was developed in order to add metadata to resources that can be physically found on the Web, such as documents. With RDF, one would describe the author of a document, its last modification time, and the subject of the document. However, RDF has been generalized to also describe objects that are not on the Web but are referenced on the Web, such as people and products. RDF describes the objects and the relationships between them and is complemented by RDF Schema and OWL, which describe the schema-level elements mentioned in RDF files. In the terminology of the previous section, RDF can be viewed as the Abox, while RDFS and OWL are the Tbox. In fact, OWL is based on description logics.

Conceptually, RDF contains statements about resources. Each statement provides a value for one of the resource's properties. A statement is a triple of the form (subject, predicate, object). As we see soon, the names of resources can get rather long. Therefore, to make the notation more palatable, we use the common practice of defining qualified names (qnames), which are shorter codes for prefixes of resources. In the example below, `ex:` is the qname for `http://www.example.org/` and `exterms:` is the qname for `http://www.example.org/terms`. Hence, `ex:movie1` is a shorthand for `http://www.example.org/movie1`.

Example 12.7

The following triples specify that the title of the movie identified by `http://www.example.org/movies/movie1` is "Life Is Beautiful" and that the subject of the review is identified by `http://www.example.org/review1.html` is the movie identified by ex:movie1. The third triple specifies the date on which the review was written.

```
ex:movie1 exterms:title "Life Is Beautiful"
ex:review1.html exterms:movieReviewed ex:movie1
ex:review1.html exterms:written-date "August 15, 2008"
```

Hence, RDF is a language for expressing unary and binary relations. The correspondence between RDF statements and Abox assertions is straightforward. RDF specifications can be (and often are) modeled as graphs. Figure 12.2 shows the graph representation of the three triples in Example 12.7. The *RDF/XML specification* shows how to serialize a set of RDF statements as XML and includes a few shortcuts that make it more parsimonious.

Example 12.8

The following XML is a serialization of three RDF statements that have the same subject, `ex:review1.html`. Note that RDF/XML enables all the properties of `ex:review1.html` to be

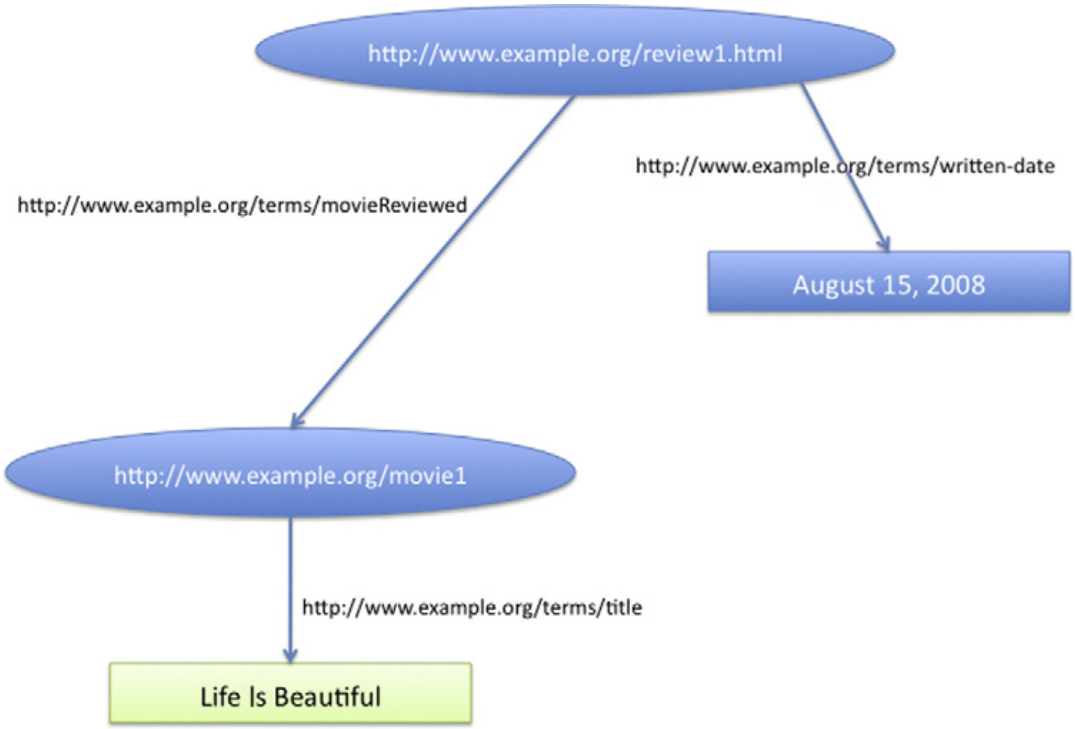

FIGURE 12.2 A graph representation of RDF triples.

nested in one XML element whose `about` value is the subject (lines 5-9), rather than having to repeat the subject three times.

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.              xmlns:dc="http://purl.org/dc/elements/1.1/"
4.              xmlns:exterms="http://www.example.org/terms/">
5.    <rdf:Description rdf:about="http://www.example.org/review1.html">
6.       <exterms:written-date>August 15, 2008</exterms:creation-date>
7.       <exterms:movieReviewed  rdf:resource="http://www.example.org/movie1/>
8.       <exterms:writtenBy  rdf:resource="http://www.example.org/staffID/23440>
9.    </rdf:Description>
10. </rdf:RDF>
```

We will not discuss all the details of RDF here, but below we explain some of its most important features and explain how it is designed to support large-scale knowledge sharing.

UNIFORM RESOURCE IDENTIFIERS

Statements in RDF include two kinds of constants: Uniform Resource Identifiers (URIs) and literals. The subject and predicate of an RDF statement are required to be URIs (except for blank nodes, which we discuss shortly) and the object can be either a URI or a literal.

A URI is a reference to a resource on the Web and therefore provides a more global mechanism for referring to entities. Consider, for example, a reference to a person. In a traditional knowledge base, we may have a string denoting her name, such as MarySmith. However, that string and even an internal unique identifier are only meaningful *within* the database or knowledge base. In RDF, we can refer to `www.example.com/staffID/5544`, and that reference is available to anyone who wants to refer to the same individual. Similarly, we can have common references to predicates, such as `www.example.com/exterms/writtenBy`, or values such as `www.example.com/exvalues/Blue`.

RDF's use of URIs is one of the crucial aspects of the language and makes it suitable for data integration and sharing. Except for providing a more global mechanism for referring to entities, it also encourages data authors to converge on a common terminology, since they can reuse existing terms. It is important to emphasize that the ability to refer to URIs does not *force* authors to confirm to standards, but *encourages* them to do so.

LITERALS IN RDF

RDF does not have its own set of built-in data types. Instead, it includes a mechanism for specifying which data type a particular literal is drawn from. These data types are also referred to by URIs, thereby creating an open type system. For example, the following statements specify the age and birth date of two staff members. The first statement specifies that the age is given as an integer, and the second specifies that the birth date is given as a date.

```
exstaff:23440  exterms:age  "42"^^xsd:integer
exstaff:54322  exterms:birthdate  "1980-05-06"^^xsd:date
```

BLANK OBJECTS

As mentioned above, RDF can represent binary relationships between objects. However, binary relations are not always sufficient. Consider representing the address of Mary from the previous example. The address is a structured object that contains multiple fields, such as street, city, country, and zip code. Representing each of these as a property of Mary is unnatural. Instead, RDF allows *blank nodes*, which are nodes without a specific identifier. In our example, shown in Figure 12.3, we would use a blank node to represent the address object of Mary, and the blank node would have the specific address properties.

Blank nodes do not have unique IDs. We can assign them IDs within an RDF document in cases where we want to refer to them more than once (e.g., if the address represents both the billing and shipping address of an order), but the IDs are meaningless across documents. Hence, if we merge two RDF graphs we need to ensure *a priori* that the IDs

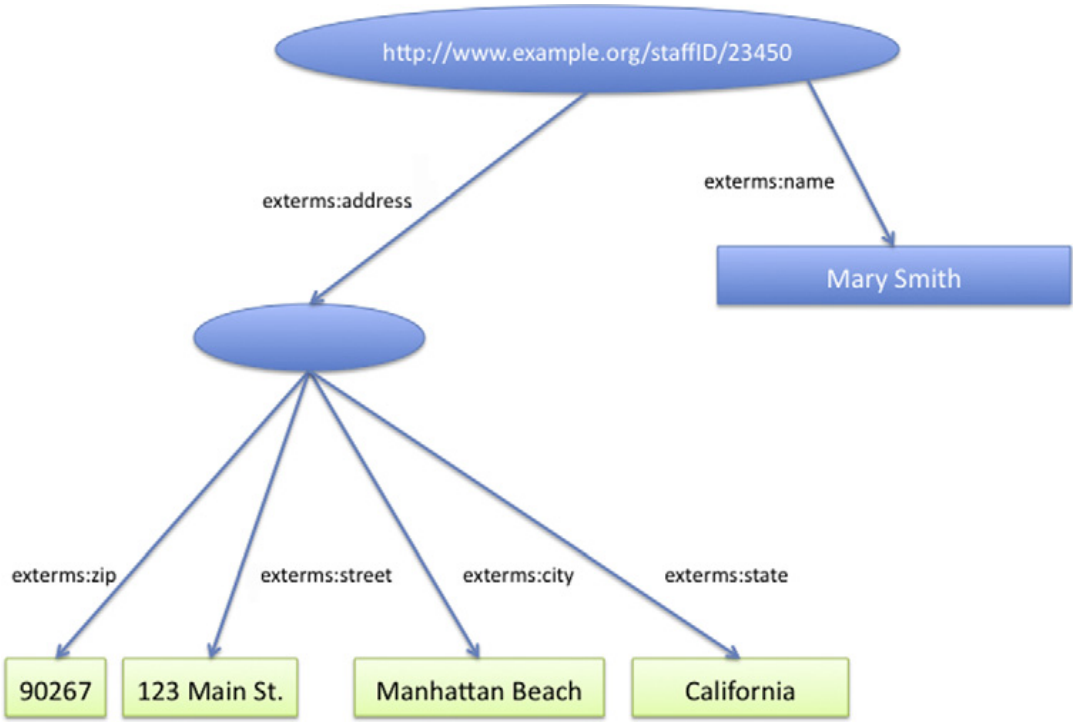

FIGURE 12.3 Blank nodes in RDF graphs.

of their blank nodes are disjoint. RDF also has constructs for representing bags, lists, and alternatives.

REIFICATION

As we will discuss in Chapter 14, it is often important to know the *provenance* of data on the Web. The provenance includes where the data came from and when (see Chapter 14). RDF provides a mechanism for reification of statements, which enables stating facts about them. Specifically, we reify an RDF triple by referring to it as a resource.

Example 12.9

The following RDF triples reify the statement about the title of `movie1`. The first statement gives the reified statement a name (`triple12345`), and the next three statements specify its properties.

```
moviedb:triple12345   rdf:type        rdf:Statement.
moviedb:triple12345   rdf:subject     ex:movie1.
moviedb:triple12345   rdf:predicate   exterms:title.
moviedb:triple12345   rdf:object      "Life Is Beautiful"
```

Given the reification, we can now assert additional facts about the statement. Below we specify who entered the triple into the movie database and when.

```
moviedb:triple12345   exterms:entered-by  ex:staffID/23340
moviedb:triple12345   exterms:entered-date "2007-07-07"^^xsd:date
```

12.3.2 RDF Schema

RDF Schema (RDFS) provides constructs for defining classes, hierarchies on classes, and membership of resources in classes. RDF Schema also allows declaring restrictions on the domains and ranges of properties. RDF Schema preceded the Web Ontology Language, which we cover shortly, and has gained adoption on its own. It is important to note that RDF does not *enforce* any of the semantics of its constructs. RDF is just a language, and it is up to the implementation of RDF systems to utilize their semantics.

The constructs of RDF Schema are themselves described by an RDF vocabulary at `http://www.w3.org/2000/01/rdf-schema#`, and its qualified name is rdfs:. The most basic construct of RDFS is `rdfs:type`, which is used to denote membership of a resource in a class. The following states that `Movie` is a class, by specifying that it has type `rdfs:class`.

```
ex:Movie rdfs:type rdfs:class
```

The same construct is used to specify instances of the class. Here, `movie1` is asserted to be a movie:

```
ex:movie1 rdfs:type ex:Movie
```

These two statements illustrate a very powerful feature of RDFS: a resource that is a class can itself be a member of another class! Hence, we can have the following scenario:

Example 12.10

The following triples state that `Corolla` is a class with instance `car11`. However, `Corolla` is also a member of the class `CarModels`.

```
ex:Corolla rdfs:type rdfs:Class
ex:car11 rdfs:type ex:Corolla
ex:CarModels rdfs:type rdfs:Class
ex:Corolla rdfs:type ex:CarModels
```

It is important to note the difference between the statements above and the scenario in which we could declare `Corolla` to be a *subclass* of `CarModels`. In the latter case, we make a clear distinction between schema elements and data elements. Individual cars are data elements, and the collection of Corollas and car models are schema. However, what if we want to describe aspects of Corollas, such as how many of them were sold in the last year? This is an attribute

of Corollas as a class, not of its individual instances. Hence, RDF offers greater modeling flexibility.

Subclasses in RDFS can be declared using the `rdfs:subclassOf` construct, as in the following:

```
ex:Comedy rdfs:subclassOf ex:Movie
```

Note that multiple statements are interpreted as conjunction. Hence, if we added

```
ex:Comedy rdfs:subclassOf ex:LaughProvokingEvents
```

then `Comedy` is asserted to be in the intersection of `Movie` and `LaughProvokingEvents`.

Finally, properties in RDFS are described using the following constructs. In the statements below we define `actor` to be a property with subproperty `leadActor`. The range of `actor` is `Person` and its domain is `Movie`.

```
exterms:actor rdfs:type rdfs:Property
exterms:leadActor rdfs:subPropertyOf exterms:actor
exterms:actor rdfs:range exterms:Person
exterms:actor rdfs:domain exterms:Movie
```

12.3.3 Web Ontology Language (OWL)

RDF Schema provides constructs that are very similar to object-oriented database schemas. OWL is inspired by the types of reasoning supported in description logics and was developed to carry such inferences over to the Semantic Web, in the spirit of the example in Section 12.1.

As described in Section 12.2, description logics are a family of languages, each with some set of constructors (and hence different computational properties). Therefore, selecting one particular description logic to be the basis for OWL would be somewhat arbitrary. Instead, the OWL standard proposes three versions: (1) OWL-Lite, which is meant to capture a very small subset of description logics but for which reasoning is very efficient, (2) OWL-DL, a version of OWL that includes a rich set of constructors found in description logics but whose computational complexity is very high, and (3) OWL-Full, which allows reification of statements in addition to the constructs of OWL-DL. Below we describe the schema-level constructs in OWL.

OWL-Lite: Unlike description logics, the OWL languages do not make the unique-names assumption. Indeed, on the Web, entities are likely to have more than one name. Hence, OWL-Lite enables specifying that two entities are the same (`sameAs`) or that they are different (`differentFrom`). It also includes the construct allDifferent for specifying that a set of objects is pairwise different from each other.

OWL-Lite is designed so that reasoning and question answering are very efficient, even in the presence of a large number of individuals. The main constructs included in OWL-Lite are the following:

- all the constructs of RDF Schema that we described above,
- some constructs from the language $\mathcal{L}$ that we described in Section 12.2, including the conjunction operator on classes ($\sqcap$), universal quantification ($\forall R.C$), and number restrictions; number restrictions are limited to only use 0 or 1,
- existential quantification on properties ($\exists R.C$); as an example, with this constructor we can describe the set of movies that have at least one actor who is female,
- a few constructs for specifying constraints on properties: `TransitiveProperty`, `SymmetricProperty`, `FunctionalProperty`, and `InverseFunctionalProperty`.

OWL-DL: OWL-DL allows the following constructs in addition to those of OWL-Lite:

- arbitrary cardinality restrictions,
- union, complement, and intersections of arbitrary concept descriptions,
- disjointness: specifying that two classes A and B are disjoint from each other,
- oneOf: describing a concept composed of an enumeration of several individuals; for example, WeekDay can be defined as oneOf(Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday),
- hasValue: describing a concept that must have a particular value as a filler on a role; for example, (nationality Vietnam) describes the set of individuals that have Vietnam as one of the fillers on their nationality role. Note that the role need not be functional.

In summary, OWL is an adaptation of description logics to the context of the Web. It adopts the use of URIs to allow more global references to terms. Like description logics, OWL is based on the open-world assumption, which is appropriate for the Web context. Finally, unlike description logics, OWL does not make the unique-names assumption, since on the Web there are likely to be many ways of referring to the same entity.

12.3.4 Querying RDF: the SPARQL Language

SparQL is a language for querying repositories of RDF triples. SparQL borrows ideas from other database query languages but provides a native RDF query language. The language is based on matching patterns of triples expressed by variables and constants, and the result of a query can be either a set of tuples or an RDF graph.

Example 12.11

Consider querying the following four RDF triples.

```
exstaff:23440  exterms:name  exstaff:person1
exstaff:23440  exterms:city  "New York"
```

```
exstaff:23444  exterms:name  exstaff:person2
exstaff:23444  exterms:city  "San Francisco"
```

The query below selects the people living in New York. Note that the variable `?d` needs to match two triples.

```
SELECT ?d ?person
WHERE
     {?d exterms:city "New York"} .
      ?d exterms:name ?person}
```

The result is a table with two columns.

?d	?person
exstaff:23440	exstaff:person1

Using `CONSTRUCT`, it is possible to create an RDF graph as output. The following outputs a graph with the `exterms:city` predicates renamed into `exterms:livesIn`.

```
CONSTRUCT {?d exterms:livesIn ?city }
WHERE
   { ?d exterms:city ?city }

exstaff:23440  exterms:livesIn  "New York"
exstaff:23444  exterms:livesIn  "San Francisco"
```

Bibliographic Notes

The application of knowledge representation techniques to data integration was investigated from the early days of the field. Catarci and Lenzerini [112] were the first to articulate how description logics can be used to model data sources and reason about relationships between them. Some systems used AI planning techniques to perform reasoning after describing the sources and mediated schema in description logics [37, 38, 361]. The Information Manifold System combined description logics with Local-as-View descriptions [380], and [60, 107] explore the complexity of answering queries using views in the presence of description logics.

The handbook of description logics [1] provides a comprehensive guide to the theory underlying description logics, the systems used to implement the logics, and some of their applications. A shorter paper that provides some of the complexity analysis of different logics is [192]. Borgida [93] describes the relevance of description logics to data management in general.

Several works have tried combining description logics with datalog. In [193] the authors explored adding unary predicates to datalog programs, where the unary predicates were

classes defined in description logics. The CARIN languages [377] extended that work by considering both unary and binary predicates in datalog rules, both of which are defined in a description logic's Tbox. In [377] the authors show tight conditions on when reasoning with a combination of description logics and datalog is decidable. Several other systems incorporated description logics into data integration and peer-data management systems, such as [11, 255, 367].

The vision of the Semantic Web was first outlined in [71]. Since then, there has been an annual conference dedicated to advances on the topic. Noy [466] surveys the approaches to semantic integration based on ontologies. Given that ontologies are more expressive than schemas, there has also been work in the community on *ontology alignment*, which has the same goal as schema mapping, but applied to ontologies. Work on this topic includes [183, 440, 464]. In fact, there are annual competitions comparing ontology alignment algorithms. The Linked Data effort [88] focuses on a subset of the Semantic Web vision. The focus is on ensuring that data sets are linked via shared identifiers, making it possible to programmatically traverse relationships across multiple data sources. The current state of the effort can be investigated at http://linkeddata.org.

Detailed specifications of the RDF, RDFS, SPARQL, and OWL standards can be found on the World Wide Web Consortium Web site at www.w3.org. There are multiple other standards related to the Semantic Web, such as OWL2 (a newer version of OWL) and the Rules Interchange Format (for specifying datalog-style and business rules). http://swoogle.umbc.edu offers a search engine over a large collection of ontologies found on the Web.