1

An Introduction to Description Logics

Daniele Nardi

Ronald J. Brachman

Abstract

This introduction presents the main motivations for the development of Description Logics (DL) as a formalism for representing knowledge, as well as some important basic notions underlying all systems that have been created in the DL tradition. In addition, we provide the reader with an overview of the entire book and some guidelines for reading it.

We first address the relationship between Description Logics and earlier semantic network and frame systems, which represent the original heritage of the field. We delve into some of the key problems encountered with the older efforts. Subsequently, we introduce the basic features of Description Logic languages and related reasoning techniques.

Description Logic languages are then viewed as the core of knowledge representation systems, considering both the structure of a DL knowledge base and its associated reasoning services. The development of some implemented knowledge representation systems based on Description Logics and the first applications built with such systems are then reviewed.

Finally, we address the relationship of Description Logics to other fields of Computer Science. We also discuss some extensions of the basic representation language machinery; these include features proposed for incorporation in the formalism that originally arose in implemented systems, and features proposed to cope with the needs of certain application domains.

1.1 Introduction

Research in the field of knowledge representation and reasoning is usually focused on methods for providing high-level descriptions of the world that can be effectively used to build intelligent applications. In this context, "intelligent" refers to the abil-

ity of a system to find implicit consequences of its explicitly represented knowledge. Such systems are therefore characterized as knowledge-based systems.

Approaches to knowledge representation developed in the 1970's—when the field enjoyed great popularity—are sometimes divided roughly into two categories: logic-based formalisms, which evolved out of the intuition that predicate calculus could be used unambiguously to capture facts about the world; and other, non-logic-based representations. The latter were often developed by building on more cognitive notions—for example, network structures and rule-based representations derived from experiments on recall from human memory and human execution of tasks like mathematical puzzle solving. Even though such approaches were often developed for specific representational chores, the resulting formalisms were usually expected to serve in general use. In other words, the non-logical systems created from very specific lines of thinking (e.g., early Production Systems) evolved to be treated as general purpose tools, expected to be applicable in different domains and on different types of problems.

On the other hand, since first-order logic provides very powerful and general machinery, logic-based approaches were more general-purpose from the very start. In a logic-based approach, the representation language is usually a variant of first-order predicate calculus, and reasoning amounts to verifying logical consequence. In the non-logical approaches, often based on the use of graphical interfaces, knowledge is represented by means of some *ad hoc* data structures, and reasoning is accomplished by similarly *ad hoc* procedures that manipulate the structures. Among these specialized representations we find *semantic networks* and *frames*. Semantic Networks were developed after the work of Quillian [1967], with the goal of characterizing by means of network-shaped cognitive structures the knowledge and the reasoning of the system. Similar goals were shared by later frame systems [Minsky, 1981], which rely upon the notion of a "frame" as a prototype and on the capability of expressing relationships between frames. Although there are significant differences between semantic networks and frames, both in their motivating cognitive intuitions and in their features, they have a strong common basis. In fact, they can both be regarded as network structures, where the structure of the network aims at representing sets of individuals and their relationships. Consequently, we use the term *network-based structures* to refer to the representation networks underlying semantic networks and frames (see [Lehmann, 1992] for a collection of papers concerning various families of network-based structures).

Owing to their more human-centered origins, the network-based systems were often considered more appealing and more effective from a practical viewpoint than the logical systems. Unfortunately they were not fully satisfactory because of their usual lack of precise semantic characterization. The end result of this was that every system behaved differently from the others, in many cases despite virtually identical-

looking components and even identical relationship names. The question then arose as to how to provide semantics to representation structures, in particular to semantic networks and frames, which carried the intuition that, by exploiting the notion of hierarchical structure, one could gain both in terms of ease of representation and in terms of the efficiency of reasoning.

One important step in this direction was the recognition that frames (at least their core features) could be given a semantics by relying on first-order logic [Hayes, 1979]. The basic elements of the representation are characterized as unary predicates, denoting sets of individuals, and binary predicates, denoting relationships between individuals. However, such a characterization does not capture the constraints of semantic networks and frames with respect to logic. Indeed, although logic is the natural basis for specifying a meaning for these structures, it turns out that frames and semantic networks (for the most part) did not require all the machinery of first-order logic, but could be regarded as fragments of it [Brachman and Levesque, 1985]. In addition, different features of the representation language would lead to different fragments of first-order logic. The most important consequence of this fact is the recognition that the typical forms of reasoning used in structure-based representations could be accomplished by specialized reasoning techniques, without necessarily requiring first-order logic theorem provers. Moreover, reasoning in different fragments of first-order logic leads to computational problems of differing complexity.

Subsequent to this realization, research in the area of Description Logics began under the label *terminological systems*, to emphasize that the representation language was used to establish the basic terminology adopted in the modeled domain. Later, the emphasis was on the set of concept-forming constructs admitted in the language, giving rise to the name *concept languages*. In more recent years, after attention was further moved towards the properties of the underlying logical systems, the term *Description Logics* became popular.

In this book we mainly use the term "Description Logics" (DL) for the representation systems, but often use the word "concept" to refer to the expressions of a DL language, denoting sets of individuals; and the word "terminology" to denote a (hierarchical) structure built to provide an intensional representation of the domain of interest.

Research on Description Logics has covered theoretical underpinnings as well as implementation of knowledge representation systems and the development of applications in several areas. This kind of development has been quite successful. The key element has been the methodology of research, based on a very close interaction between theory and practice. On the one hand, there are various implemented systems based on Description Logics, which offer a palette of description formalisms with differing expressive power, and which are employed in various application do-

mains (such as natural language processing, configuration of technical products, or databases). On the other hand, the formal and computational properties of reasoning (like decidability and complexity) of various description formalisms have been investigated in detail. The investigations are usually motivated by the use of certain constructors in implemented systems or by the need for these constructors in specific applications—and the results have influenced the design of new systems.

This book is meant to provide a thorough introduction to Description Logics, covering all the above-mentioned aspects of DL research—namely theory, implementation, and applications. Consequently, the book is divided into three parts:

- Part I introduces the theoretical foundations of Description Logics, addressing some of the most recent developments in theoretical research in the area;
- Part II focuses on the implementation of knowledge representation systems based on Description Logics, describing the basic functionality of a DL system, surveying the most influential knowledge representation systems based on Description Logics, and addressing specialized implementation techniques;
- Part III addresses the use of Description Logics and of DL-based systems in the design of several applications of practical interest.

In the remainder of this introductory chapter, we review the main steps in the development of Description Logics, and introduce the main issues that are dealt with later in the book, providing pointers for its reading. In particular, in the next section we address the origins of Description Logics and then we review knowledge representation systems based on Description Logics, the main applications developed with Description Logics, the main extensions to the basic DL framework and relationships with other fields of Computer Science.

1.2 From networks to Description Logics

In this section we begin by recalling approaches to representing knowledge that were developed before research on Description Logics began (i.e., semantic networks and frames). We then provide a very brief introduction to the basic elements of these approaches, based on Tarski-style semantics. Finally, we discuss the importance of computational analyses of the reasoning methods developed for Description Logics, a major ingredient of research in this field.

1.2.1 Network-based representation structures

In order to provide some intuition about the ideas behind representations of knowledge in network form, we here speak in terms of a generic network, avoiding references to any particular system. The elements of a network are *nodes* and *links*.

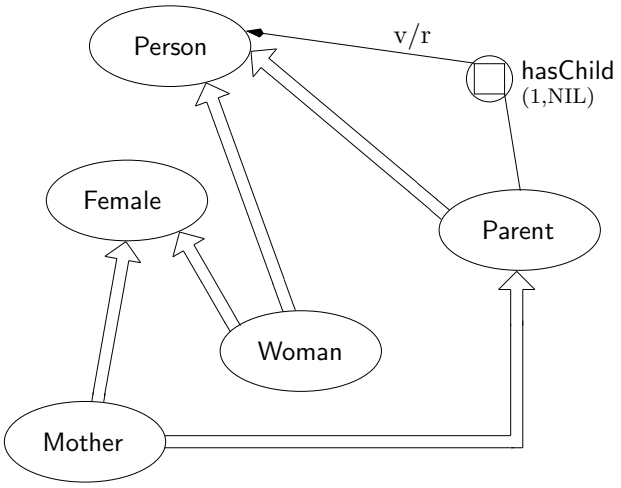

Fig. 1.1. An example network.

Typically, nodes are used to characterize concepts, i.e., sets or classes of individual objects, and links are used to characterize relationships among them. In some cases, more complex relationships are themselves represented as nodes; these are carefully distinguished from nodes representing concepts. In addition, concepts can have simple properties, often called attributes, which are typically attached to the corresponding nodes. Finally, in many of the early networks both individual objects and concepts were represented by nodes. Here, however, we restrict our attention to knowledge about concepts and their relationships, deferring for now treatment of knowledge about specific individuals.

Let us consider a simple example, whose pictorial representation is given in Figure 1.1, which represents knowledge concerning persons, parents, children, etc. The structure in the figure is also referred to as a *terminology*, and it is indeed meant to represent the generality/specificity of the concepts involved. For example the link between Mother and Parent says that "mothers are parents"; this is sometimes called an "IS-A" relationship.

The IS-A relationship defines a hierarchy over the concepts and provides the basis for the "inheritance of properties": when a concept is more specific than some other concept, it inherits the properties of the more general one. For example, if a person has an age, then a mother has an age, too. This is the typical setting of the so-called (monotonic) *inheritance networks* (see [Brachman, 1979]).

A characteristic feature of Description Logics is their ability to represent other kinds of relationships that can hold between concepts, beyond IS-A relationships. For example, in Figure 1.1, which follows the notation of [Brachman and Schmolze, 1985], the concept of Parent has a property that is usually called a "role," expressed

by a link from the concept to a node for the role labeled hasChild. The role has what is called a "value restriction," denoted by the label v/r, which expresses a limitation on the range of types of objects that can fill that role. In addition, the node has a number restriction expressed as (1,NIL), where the first number is a lower bound on the number of children and the second element is the upper bound, and NIL denotes infinity. Overall, the representation of the concept of Parent here can be read as "A parent is a person having at least one child, and all of his/her children are persons."

Relationships of this kind are inherited from concepts to their subconcepts. For example, the concept Mother, i.e., a female parent, is a more specific descendant of both the concepts Female and Parent, and as a result inherits from Parent the link to Person through the role hasChild; in other words, Mother inherits the restriction on its hasChild role from Parent.

Observe that there may be implicit relationships between concepts. For example, if we define Woman as the concept of a female person, it is the case that every Mother is a Woman. It is the task of the knowledge representation system to find implicit relationships such as these (many are more complex than this one). Typically, such inferences have been characterized in terms of properties of the network. In this case one might observe that both Mother and Woman are connected to both Female and Person, but the path from Mother to Person includes a node Parent, which is more specific then Person, thus enabling us to conclude that Mother is more specific than Person.

However, the more complex the relationships established among concepts, the more difficult it becomes to give a precise characterization of what kind of relationships can be computed, and how this can be done without failing to recognize some of the relationships or without providing wrong answers.

1.2.2 A logical account of network-based representation structures

Building on the above ideas, a number of systems were implemented and used in many kinds of applications. As a result, the need emerged for a precise characterization of the meaning of the structures used in the representations and of the set of inferences that could be drawn from those structures.

A precise characterization of the meaning of a network can be given by defining a language for the elements of the structure and by providing an interpretation for the strings of that language. While the syntax may have different flavors in different settings, the semantics is typically given as a Tarski-style semantics.

For the syntax we introduce a kind of abstract language, which resembles other logical formalisms. The basic step of the construction is provided by two disjoint alphabets of symbols that are used to denote *atomic concepts*, designated by unary

predicate symbols, and *atomic roles*, designated by binary predicate symbols; the latter are used to express relationships between concepts.

Terms are then built from the basic symbols using several kinds of constructors. For example, *intersection of concepts*, which is denoted $C \sqcap D$, is used to restrict the set of individuals under consideration to those that belong to both C and D. Notice that, in the syntax of Description Logics, concept expressions are variable-free. In fact, a concept expression denotes the set of all individuals satisfying the properties specified in the expression. Therefore, $C \sqcap D$ can be regarded as the first-order logic sentence, $C(x) \wedge D(x)$, where the variable ranges over all individuals in the interpretation domain and $C(x)$ is true for those individuals that belong to the concept C.

In this book, we will present other syntactic notations that are more closely related to the concrete syntax adopted by implemented DL systems, and which are more suitable for the development of applications. One example of concrete syntax proposed in [Patel-Schneider and Swartout, 1993] is based on a LISP-like notation, where the concept of female persons, for example, is denoted by `(and Person Female)`.

The key characteristic features of Description Logics reside in the constructs for establishing relationships between concepts. The basic ones are *value restrictions*. For example, a value restriction, written $\forall R.C$, requires that all the individuals that are in the relationship R with the concept being described belong to the concept C (technically, it is all individuals that are in the relationship R with an individual described by the concept in question that are themselves describable as C's).

As for the semantics, concepts are given a set-theoretic interpretation: a concept is interpreted as a set of individuals and roles are interpreted as sets of pairs of individuals. The domain of interpretation can be chosen arbitrarily, and it can be infinite. The non-finiteness of the domain and the *open-world assumption* are distinguishing features of Description Logics with respect to the modeling languages developed in the study of databases (see Chapters 4, and 16).

Atomic concepts are thus interpreted as subsets of the intepretation domain, while the semantics of the other constructs is then specified by defining the set of individuals denoted by each construct. For example, the concept $C \sqcap D$ is the set of individuals obtained by intersecting the sets of individuals denoted by C and D, respectively. Similarly, the interpretation of $\forall R.C$ is the set of individuals that are in the relationship R with individuals belonging to the set denoted by the concept C.

As an example, let us suppose that Female, Person, and Woman are atomic concepts and that hasChild and hasFemaleRelative are atomic roles. Using the operators *intersection*, *union* and *complement* of concepts, interpreted as set operations, we can describe the concept of "persons that are not female" and the concept of "in-

dividuals that are female or male" by the expressions

$$\mathsf{Person} \sqcap \neg \mathsf{Female} \qquad \text{and} \qquad \mathsf{Female} \sqcup \mathsf{Male}.$$

It is worth mentioning that intersection, union, and complement of concepts have been also referred to as *concept conjunction*, *concept disjunction* and *concept negation*, respectively, to emphasize the relationship to logic.

Let us now turn our attention to role restrictions by looking first at quantified role restrictions and, subsequently, at what we call "number restrictions." Most languages provide *(full) existential quantification* and *value restriction* that allow one to describe, for example, the concept of "individuals having a female child" as $\exists \mathsf{hasChild}.\mathsf{Female}$, and to describe the concept of "individuals all of whose children are female" by the concept expression $\forall \mathsf{hasChild}.\mathsf{Female}$. In order to distinguish the function of each concept in the relationship, the individual object that corresponds to the second argument of the role viewed as a binary predicate is called a *role filler*. In the above expressions, which describe the properties of parents having female children, individual objects belonging to the concept Female are the fillers of the role hasChild.

Existential quantification and value restrictions are thus meant to characterize relationships between concepts. In fact, the role link between Parent and Person in Figure 1.1 can be expressed by the concept expression

$$\exists \mathsf{hasChild}.\mathsf{Person} \sqcap \forall \mathsf{hasChild}.\mathsf{Person}.$$

Such an expression therefore characterizes the concept of Parent as the set of individuals having at least one filler of the role hasChild belonging to the concept Person; moreover, every filler of the role hasChild must be a person.

Finally, notice that in quantified role restrictions the variable being quantified is not explicitly mentioned. The corresponding sentence in first-order logic is $\forall y.R(x,y) \supset C(y)$, where x is again a free variable ranging over the interpretation domain.

Another important kind of role restriction is given by *number restrictions*, which restrict the cardinality of the sets of fillers of roles. For instance, the concept

$$(\geqslant 3\, \mathsf{hasChild}) \sqcap (\leqslant 2\, \mathsf{hasFemaleRelative})$$

represents the concept of "individuals having at least three children and at most two female relatives." Number restrictions are sometimes viewed as a distinguishing feature of Description Logics, although one can find some similar constructs in some database modeling languages (notably Entity-Relationship models).

Beyond the constructs to form concept expressions, Description Logics provide constructs for roles, which can, for example, establish role hierarchies. However,

the use of role expressions is generally limited to expressing relationships between concepts.

Intersection of roles is an example of a role-forming construct. Intuitively, $\mathsf{hasChild} \sqcap \mathsf{hasFemaleRelative}$ yields the role "has-daughter," so that the concept expression

$$\mathsf{Woman} \sqcap \leqslant 2\,(\mathsf{hasChild} \sqcap \mathsf{hasFemaleRelative})$$

denotes the concept of "a woman having at most 2 daughters".

A more comprehensive view of the basic definitions of DL languages will be given in Chapter 2.

1.2.3 Reasoning

The basic inference on concept expressions in Description Logics is *subsumption*, typically written as $C \sqsubseteq D$. Determining subsumption is the problem of checking whether the concept denoted by D (the *subsumer*) is considered more general than the one denoted by C (the *subsumee*). In other words, subsumption checks whether the first concept always denotes a subset of the set denoted by the second one.

For example, one might be interested in knowing whether $\mathsf{Woman} \sqsubseteq \mathsf{Mother}$. In order to verify this kind of relationship one has in general to take into account the relationships defined in the terminology. As we explain in the next section, under appropriate restrictions, one can embody such knowledge directly in concept expressions, thus making subsumption over concept expressions the basic reasoning task. Another typical inference on concept expressions is concept *satisfiability*, which is the problem of checking whether a concept expression does not necessarily denote the empty concept. In fact, concept satisfiability is a special case of subsumption, with the subsumer being the empty concept, meaning that a concept is not satisfiable.

Although the meaning of concepts had already been specified with a logical semantics, the design of inference procedures in Description Logics was influenced for a long time by the tradition of semantic networks, where concepts were viewed as nodes and roles as links in a network. Subsumption between concept expressions was recognized as the key inference and the basic idea of the earliest subsumption algorithms was to transform two input concepts into labeled graphs and test whether one could be embedded into the other; the embedded graph would correspond to the more general concept (the subsumer) [Lipkis, 1982]. This method is called *structural comparison*, and the relation between concepts being computed is called *structural subsumption*. However, a careful analysis of the algorithms for structural subsumption shows that they are *sound*, but not always *complete* in terms of the logical semantics: whenever they return "yes" the answer is correct, but when they

report "no" the answer may be incorrect. In other words, structural subsumption is in general weaker than logical subsumption.

The need for complete subsumption algorithms is motivated by the fact that in the usage of knowledge representation systems it is often necessary to have a guarantee that the system has not failed in verifying subsumption. Consequently, new algorithms for computing subsumption have been devised that are no longer based on a network representation, and these can be proven to be complete. Such algorithms have been developed by specializing classical settings for deductive reasoning to the DL subsets of first-order logics, as done for tableau calculi by Schmidt-Schauß and Smolka [1991], and also by more specialized methods.

In the paper "The Tractability of Subsumption in Frame-Based Description Languages," Brachman and Levesque [1984] argued that there is a tradeoff between the expressiveness of a representation language and the difficulty of reasoning over the representations built using that language. In other words, the more expressive the language, the harder the reasoning. They also provided a first example of this tradeoff by analyzing the language $\mathcal{FL}^-$ (Frame Language), which included intersection of concepts, value restrictions and a simple form of existential quantification. They showed that for such a language the subsumption problem could be solved in polynomial time, while adding a construct called role restriction to the language makes subsumption a coNP-hard problem (the extended language was called $\mathcal{FL}$).

The paper by Brachman and Levesque introduced at least two new ideas:

(i) "efficiency of reasoning" over knowledge structures can be studied using the tools of computational complexity theory;
(ii) different combinations of constructs can give rise to languages with different computational properties.

An immediate consequence of the above observations is that one can study formally and methodically the tradeoff between the computational complexity of reasoning and the expressiveness of the language, which itself is defined in terms of the constructs that are admitted in the language. After the initial paper, a number of results on this tradeoff for concept languages were obtained (see Chapters 2 and 3), and these results allow us to draw a fairly complete picture of the complexity of reasoning for a wide class of concept languages. Moreover, the problem of finding the optimal tradeoff, namely the most expressive extensions of $\mathcal{FL}^-$ with respect to a given set of constructs that still keep subsumption polynomial, has been studied extensively [Donini *et al.*, 1991b; 1999].

One of the assumptions underlying this line of research is to use worst-case complexity as a measure of the efficiency of reasoning in Description Logics (and more generally in knowledge representation formalisms). Such an assumption has some-

times been criticized (see for example [Doyle and Patil, 1991]) as not adequately characterizing system performance or accounting for more average-case behavior. While this observation suggests that computational complexity alone may not be sufficient for addressing performance issues, research on the computational complexity of reasoning in Description Logics has most definitely led to a much deeper understanding of the problems arising in implementing reasoning tools. Let us briefly address some of the contributions of this body of work.

First of all, the study of the computational complexity of reasoning in Description Logics has led to a clear understanding of the properties of the language constructs and their interaction. This is not only valuable from a theoretical viewpoint, but gives insight to the designer of deduction procedures, with clear indications of the language constructs and their combinations that are difficult to deal with, as well as general methods to cope with them.

Secondly, the complexity results have been obtained by exploiting a general technique for satisfiability-checking in concept languages, which relies on a form of tableau calculus [Schmidt-Schauß and Smolka, 1991]. Such a technique has proved extremely useful for studying both the correctness and the complexity of the algorithms. More specifically, it provides an algorithmic framework that is parametric with respect to the language constructs. The algorithms for concept satisfiability and subsumption obtained in this way have also led directly to practical implementations by application of clever control strategies and optimization techniques. The most recent knowledge representation systems based on Description Logics adopt tableau calculi [Horrocks, 1998b].

Thirdly, the analysis of pathological cases in this formal framework has led to the discovery of incompleteness in the algorithms developed for implemented systems. This has also consequently proven useful in the definition of suitable test sets for verifying implementations. For example, the comparison of implemented systems (see for example [Baader *et al.*, 1992b; Heinsohn *et al.*, 1992]) has greatly benefitted from the results of the complexity analysis.

The basic reasoning techniques for Description Logics are presented in Chapter 2, while a detailed analysis of the complexity of reasoning problems in several languages is developed in Chapter 3.

After the tradeoff between expressiveness and tractability of reasoning was thoroughly analyzed and the range of applicability of the corresponding inference techniques had been experimented with, there was a shift of focus in the theoretical research on reasoning in Description Logics. Interest grew in relating Description Logics to the modeling languages used in database management. In addition, the discovery of strict relationships with expressive modal logics stimulated the study of so-called *very expressive* Description Logics. These languages, besides admitting very general mechanisms for defining concepts (for example cyclic definitions,

addressed in the next section), provide a richer set of concept-forming constructs and constructs for forming complex role expressions. For these languages, the expressiveness is great enough that the new challenge became enriching the language while retaining the decidability of reasoning. It is worth pointing out that this new direction of theoretical research was accompanied by a corresponding shift in the implementation of knowledge representation systems based on very expressive DL languages. The study of reasoning methods for very expressive Description Logics is addressed in Chapter 5.

1.3 Knowledge representation in Description Logics

In the previous section a basic representation language for Description Logics was introduced along with some key associated reasoning techniques. Our goal now is to illustrate how Description Logics can be useful in the design of knowledge-based applications, that is to say, how a DL language is used in a knowledge representation system that provides a language for defining a knowledge base and tools to carry out inferences over it. The realization of knowledge systems involves two primary aspects. The first consists in providing a precise characterization of a knowledge base; this involves precisely characterizing the type of knowledge to be specified to the system as well as clearly defining the reasoning services the system needs to provide—the kind of questions that the system should be able to answer. The second aspect consists in providing a rich development environment where the user can benefit from different services that can make his/her interaction with the system more effective. In this section we address the logical structure of the knowledge base, while the design of systems and tools for the development of applications is addressed in the next section.

One of the products of some important historical efforts to provide precise characterizations of the behavior of semantic networks and frames was a *functional approach* to knowledge representation [Levesque, 1984]. The idea was to give a precise specification of the functionality to be provided by a knowledge base and, specifically, of the inferences performed by the knowledge base—independent of any implementation. In practice, the functional description of a reasoning system is productively specified through a so-called "Tell&Ask" interface. Such an interface specifies operations that enable knowledge base construction (Tell operations) and operations that allow one to get information out of the knowledge base (Ask operations). In the following we shall adopt this view for characterizing both the definition of a DL knowledge base and the deductive services it provides.

Within a knowledge base one can see a clear distinction between *intensional knowledge*, or general knowledge about the problem domain, and *extensional knowledge*, which is specific to a particular problem. A DL knowledge base is analogously

typically comprised by two components—a "*TBox*" and an "*ABox*." The TBox contains intensional knowledge in the form of a terminology (hence the term "TBox," but "taxonomy" could be used as well) and is built through declarations that describe general properties of concepts. Because of the nature of the subsumption relationships among the concepts that constitute the terminology, TBoxes are usually thought of as having a lattice-like structure; this mathematical structure is entailed by the subsumption relationship—it has nothing to do with any implementation. The ABox contains extensional knowledge—also called assertional knowledge (hence the term "ABox")—knowledge that is specific to the individuals of the domain of discourse. Intensional knowledge is usually thought not to change—to be "timeless," in a way—and extensional knowledge is usually thought to be contingent, or dependent on a single set of circumstances, and therefore subject to occasional or even constant change.

In the rest of the section we present a basic Tell&Ask interface by analyzing the TBox and the ABox of a DL knowledge base.

1.3.1 The TBox

One key element of a DL knowledge base is given by the operations used to build the terminology. Such operations are directly related to the forms and the meaning of the declarations allowed in the TBox.

The basic form of declaration in a TBox is a concept *definition*, that is, the definition of a new concept in terms of other previously defined concepts. For example, a woman can be defined as a female person by writing this declaration:

$$\mathsf{Woman} \equiv \mathsf{Person} \sqcap \mathsf{Female}$$

Such a declaration is usually interpreted as a logical equivalence, which amounts to providing both sufficient and necessary conditions for classifying an individual as a woman. This form of definition is much stronger than the ones used in other kinds of representations of knowledge, which typically impose only necessary conditions; the strength of this kind of declaration is usually considered a characteristic feature of DL knowledge bases. In DL knowledge bases, therefore, a terminology is constituted by a set of concept definitions of the above form.

However, there are some important common assumptions usually made about DL terminologies:

- only one definition for a concept name is allowed;
- definitions are *acyclic* in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

This kind of restriction is common to many DL knowledge bases and implies that

every defined concept can be expanded in a unique way into a complex expression containing only atomic concepts by replacing every defined concept with the right-hand side of its definition.

Nebel [1990b] showed that even simple expansion of definitions like this gives rise to an unavoidable source of complexity; in practice, however, definitions that inordinately increase the complexity of reasoning do not seem to occur. Under these assumptions the computational complexity of inferences can be studied by abstracting from the terminology and by considering all given concepts as fully expanded expressions. Therefore, much of the study of reasoning methods in Description Logics has been focused on concept expressions and, more specifically, as discussed in the previous section, on subsumption, which can be considered the basic reasoning service for the TBox.

In particular, the basic task in constructing a terminology is *classification*, which amounts to placing a new concept expression in the proper place in a taxonomic hierarchy of concepts. Classification can be accomplished by verifying the subsumption relation between each defined concept in the hierarchy and the new concept expression. The placement of the concept will be in between the most specific concepts that subsume the new concept and the most general concepts that the new concept subsumes.

More general settings for concept definitions have recently received some attention, deriving from attempts to establish formal relationships between Description Logics and other formalisms and from attempts to satisfy a need for increased expressive power. In particular, the admission of cyclic definitions has led to different semantic interpretations of the declarations, known as greatest/least fixed-point, and descriptive semantics. Although it has been argued that different semantics may be adopted depending on the target application, the more commonly adopted one is descriptive semantics, which simply requires that all the declarations be satisfied in the interpretation. Moreover, by dropping the requirement that on the left-hand side of a definition there can only be an atomic concept name, one can consider so-called *(general) inclusion axioms* of the form

$$C \sqsubseteq D$$

where C and D are arbitrary concept expressions. Notice that a concept definition can be expressed by two general inclusions. As a result of several theoretical studies concerning both the decidability of and implementation techniques for cyclic TBoxes, the most recent DL systems admit rather powerful constructs for defining concepts.

The basic deduction service for such TBoxes can be viewed as *logical implication* and it amounts to verifying whether a generic relationship (for example a subsumption relationship between two concept expressions) is a logical consequence of the

declarations in the TBox. The issues arising in the semantic characterization of cyclic TBoxes are dealt with in Chapter 2, while techniques for reasoning in cyclic TBoxes are addressed also in Chapter 2 and in Chapter 5, where very expressive Description Logics are presented.

1.3.2 The ABox

The ABox contains extensional knowledge about the domain of interest, that is, assertions about individuals, usually called *membership assertions*. For example,

$$\mathsf{Female} \sqcap \mathsf{Person}(\mathsf{ANNA})$$

states that the individual ANNA is a female person. Given the above definition of woman, one can derive from this assertion that ANNA is an instance of the concept Woman. Similarly,

$$\mathsf{hasChild}(\mathsf{ANNA}, \mathsf{JACOPO})$$

specifies that ANNA has JACOPO as a child. Assertions of the first kind are also called *concept assertions*, while assertions of the second kind are also called *role assertions*.

As illustrated by these examples, in the ABox one can typically specify knowledge in the form of concept assertions and role assertions. In concept assertions general concept expressions are typically allowed, while role assertions, where the role is not a primitive role but a role expression, are typically not allowed, being treated in the case of very expressive languages only.

The basic reasoning task in an ABox is *instance checking*, which verifies whether a given individual is an instance of (belongs to) a specified concept. Although other reasoning services are usually considered and employed, they can be defined in terms of instance checking. Among them we find *knowledge base consistency*, which amounts to verifying whether every concept in the knowledge base admits at least one individual; *realization*, which finds the most specific concept an individual object is an instance of; and *retrieval*, which finds the individuals in the knowledge base that are instances of a given concept. These can all be accomplished by means of instance checking.

The presence of individuals in a knowledge base makes reasoning more complex from a computational viewpoint [Donini *et al.*, 1994b], and may require significant extensions of some TBox reasoning techniques. Reasoning in the ABox is addressed in Chapter 3.

It is worth emphasizing that, although we have separated out for convenience the services for the ABox, when the TBox cannot be dealt with by means of the simple substitution mechanism used for acyclic TBoxes, the reasoning services may have to

take into account all of the knowledge base including both the TBox and the ABox, and the corresponding reasoning problems become more complex. A full setting including general TBox and ABox is addressed in Chapter 5, where very expressive Description Logics are discussed.

More general languages for defining ABoxes have also been considered. Knowledge representation systems providing a powerful logical language for the ABox and a DL language for the TBox are often considered *hybrid* reasoning systems, since completely different knowledge representation languages may be used to specify the knowledge in the different components. Hybrid reasoning systems were popular in the 1980's (see for example [Brachman *et al.*, 1985]); lately, the topic has regained attention [Levy and Rousset, 1997; Donini *et al.*, 1998b], focusing on knowledge bases with a DL component for concept definitions and a logic-programming component for assertions about individuals. Sound and complete inference methods for hybrid knowledge bases become difficult to devise whenever there is a strict interaction between the knowledge components.

1.4 From theory to practice: Description Logics systems

A direct practical result of research on knowledge representation has been the development of tools for the construction of knowledge-based applications. As already noted, research on Description Logics has been characterized by a tight connection between theoretical results and implementation of systems. This has been achieved by maintaining a very close relationship between theoreticians, system implementors and users of knowledge representation systems based on Description Logics (DL-KRS). The results of work on reasoning algorithms and their complexity has influenced the design of systems, and research on reasoning algorithms has itself been focused by a careful analysis of the capabilities and the limitations of implemented systems. In this section we first sketch the functionality of some knowledge representation systems and, subsequently, discuss the evolution of DL-KRS. The reader can find a deeper treatment of the first topic in Chapter 7, while a survey of knowledge representation systems based on Description Logics is provided in Chapter 8. Chapter 9 is devoted to more specialized implementation and optimization techniques.

1.4.1 The design of knowledge representation systems based on Description Logics

In order to appreciate the difficulties of implementing and maintaining a knowledge representation system, it is necessary to consider that in the usage of a knowledge representation system, the reasoning service is really only one aspect of a complex

system, one which may even be hidden from the final user. The user, before getting to "push the reasoning button," has to model the domain of interest, and input knowledge into the system. Further, in many cases, a simple yes/no answer is of little use, so a simplistic implementation of the Tell&Ask paradigm may be inadequate. As a consequence, the path one follows to get from the identification of a suitable knowledge representation system to the design of applications based on it is a complex and demanding one (see for example [Brachman, 1992]). In the case of Description Logics, this is especially true if the goal is to devise a system to be used by users who are not DL experts and who need to obtain a working system as quickly as possible. In the 1980's, when frame-based systems (such as, for example, KEE [Fikes and Kehler, 1985]; see [Karp, 1992] for an overview) had reached the strength of commercial products, the burden on a user of moving to the more modern DL-KRS had to be kept small. Consequently, a stream of research addressed important aspects of the pragmatic usability of DL systems. This issue was especially relevant for those systems aiming at limiting the expressiveness of the language, but providing the user with sound, complete and efficient reasoning services. The issue of embedding a DL language within an environment suitable for application development is further addressed in Chapter 7.

In recent years, we might add, useful DL systems have often come as internal components of larger environments whose interfaces could completely hide the DL language and its core reasoning services. Systems like IMACS [Brachman *et al.*, 1993] and PROSE [Wright *et al.*, 1993] were quite successful in classifying data and configuring products, respectively, without the need for any user to understand the details of the DL representation language (CLASSIC) they were built upon.

Nowadays, applications for gathering information from the World-Wide Web, where the interface can be specifically designed to support the retrieval of such information, also hide the knowledge representation and reasoning component. In addition, some data modeling tools, where the system provides a more conventional interface, can provide additional facilities based on the capability of reasoning about models with a DL inference engine. The possible settings for taking advantage of Description Logics as components of larger systems are discussed in Part III; more specifically, Chapter 14 presents Web applications and Chapter 15 Natural Language applications, while the reasoning capabilities of Description Logics in Database applications are addressed in Chapter 16.

1.4.2 Knowledge representation systems based on Description Logics

The history of knowledge representation is covered in the literature in numerous ways (see for example [Woods and Schmolze, 1992; Rich, 1991; Baader *et al.*, 1992b]). Here we identify three generations of systems, highlighting their historical

evolution rather than their specific functionality. We shall characterize them as *Pre-DL systems*, *DL systems* and *Current Generation DL systems*. Detailed references to implemented systems are given in Chapter 8.

1.4.2.1 Pre-Description Logics systems

The ancestor of DL systems is KL-ONE [Brachman and Schmolze, 1985], which signaled the transition from semantic networks to more well-founded terminological (description) logics. The influence of KL-ONE was profound and it is considered the root of the entire family of languages [Woods and Schmolze, 1990].

Semantic networks were introduced around 1966 as a representation for the concepts underlying English words, and became a popular type of framework for representing a wide variety of concepts in AI applications. Important and commonsensical ideas evolved in this work, from named nodes and links for representing concepts and relationships, to hierarchical networks with inheritance of properties, to the notion of "instantiation" of a concept by an individual object. But semantic network systems were fraught with problems, including vagueness and inconsistency in the meaning of various constructs, and the lack of a level of structure on which to base application-independent inference procedures. In his Ph.D. thesis [Brachman, 1977a] and subsequent work (e.g., see [Brachman, 1979]), Brachman addressed representation at what he called an "epistemological," or knowledge-structuring level. This led to a set of primitives for structuring knowledge that was less application- and world-knowledge-dependent than "semantic" representations (like those for processing natural language case structures), yet richer than the impoverished set of primitives available in strictly logical languages. The main result of this work was a new knowledge representation framework whose primitive elements allowed cleaner, more application-independent representations than prior network formalisms. In the late 1970's, Brachman and his colleagues explored the utility and implications of this kind of framework in the KL-ONE system.

KL-ONE introduced most of the key notions explored in the extensive work on Description Logics that followed. These included, for example, the notions of concepts and roles and how they were to be interrelated; the important ideas of "value restriction" and "number restriction," which modified the use of roles in the definitions of concepts; and the crucial inferences of subsumption and classification. It also sowed the seeds for the later distinction between the TBox and ABox and a host of other significant notions that greatly influenced subsequent work. KL-ONE also was the initial example of the substantial interplay between theory and practice that characterizes the history of Description Logics. It was influenced by work in logic and philosophy (and in turn itself influenced work in philosophy and psychology), and significant care was taken in its design to allow it to be consistent and semantically sound. But it was also used in multiple applications, covering intel-

ligent information presentation and natural language understanding, among other things.

Most of the focus of the original work on KL-ONE was on the representation of and reasoning with concepts, with only a small amount of attention paid to reasoning with individual objects. The first descendants of KL-ONE were focused on architectures providing a clear distinction between a powerful logic-based (or rule-based) component and a specialized terminological component. These systems came to be referred to as *hybrid systems*. A major research issue was the integration of the two components to provide unified reasoning services over the whole knowledge base.

1.4.2.2 Description Logics systems

The earliest "pre-DL" systems derived directly from KL-ONE, which, while itself a direct result of formal analysis of the shortcomings of semantic networks, was mainly about the implementation of a viable classification algorithm and the data structures to adequately represent concepts. Description Logic systems, *per se*, which followed as the next generation, were more derived from a wave of theoretical research on terminological logics that resulted from examination of KL-ONE and some other early systems. This work was initiated in roughly 1984, inspired by a paper by Brachman and Levesque [Brachman and Levesque, 1984] on the formal complexity of reasoning in Description Logics. Subsequent results on the trade-off between the expressiveness of a DL language and the complexity of reasoning with it, and more generally, the identification of the sources of complexity in DL systems, showed that a careful selection of language constructs was needed and that the reasoning services provided by the system are deeply influenced by the set of constructs provided to the user. We can thus characterize three different approaches to the implementation of reasoning services. The first can be referred to as *limited+complete*, and includes systems that are designed by restricting the set of constructs in such a way that subsumption would be computed efficiently, possibly in polynomial time. The CLASSIC system [Brachman *et al.*, 1991] is the most significant example of this kind. The second approach can be denoted as *expressive+incomplete*, since the idea is to provide both an expressive language and efficient reasoning. The drawback is, however, that reasoning algorithms turn out to be incomplete in these systems. Notable examples of this kind of system are LOOM [MacGregor and Bates, 1987], and BACK [Nebel and von Luck, 1988]. After some of the sources of incompleteness were discovered, often by identifying the constructs—or, more precisely, combinations of constructs—that would require an exponential algorithm to preserve the completeness of reasoning, systems with complete reasoning algorithms were designed. Systems of this sort (see for example KRIS [Baader and Hollunder, 1991a]) are therefore characterized as *expres-*

sive+complete; they were not as efficient as those following the other approaches, but they provided a testbed for the implementation of reasoning techniques developed in the theoretical investigations, and they played an important role in stimulating comparison and benchmarking with other systems [Heinsohn *et al.*, 1992; Baader *et al.*, 1992b].

1.4.2.3 Current generation Description Logics systems

In the current generation of DL-KRS, the need for complete algorithms for expressive languages has been the focus of attention. The expressiveness of the DL language required for reasoning on data models and semi-structured data has contributed to the identification of the most important extensions for practical applications.

The design of complete algorithms for expressive Description Logics has led to significant extensions of tableau-based techniques and to the introduction of several optimization techniques, partly borrowed from theorem proving and partly specifically developed for Description Logics. The first example of a system developed along these lines is FaCT [Horrocks, 1998b].

This research has also been influenced by newly discovered relationships between Description Logics and other logics, leading to exchanging benchmarks and experimental comparisons with other deduction systems.

The techniques that have been used in the implementation of very expressive Description Logics are addressed in detail in Chapter 9.

1.5 Applications developed with Description Logics systems

The third component in the picture of the development of Description Logics is the implementation of applications in different domains. Some of the applications created over the years may have only reached the level of prototype, but many of them have the completeness of industrial systems and have been deployed in production use.

A critical element in the development of applications based on Description Logics is the usability of the knowledge representation system. We have already emphasized that building a tool to be used in the design and implementation of knowledge-based applications requires significant work to make it suitable for interactive development, explanation and debugging, interface implementation, and so on. In addition, here we focus on the effectiveness of Description Logics as a modeling language. A modeling language should have intuitive semantics and the syntax must help convey the intended meaning. To this end, a somewhat different syntax than we have seen so far, closer to that of natural language, has often been adopted, and graphical interfaces that provide an operational view of the process of knowledge

base construction have been developed. The issues arising in modeling application domains using Description Logics are dealt with in Chapter 10, and will be briefly addressed in the next subsection.

It is natural to expect that some classes of applications share similarities both in methodological patterns and in the design of specific structures or reasoning capabilities. Consequently, we identify several application domains in Section 1.5.2; these include Software Engineering, Configuration, Medicine, and Digital Libraries and Web-based Information Systems.

In Section 1.5.3 we consider several application areas where Description Logics play a major role; these include Natural Language Processing as well as Database Management, where Description Logics can be used in several ways.

When addressing the design of applications it is also worth pointing out that there has been significant evolution in the way Description Logics have been used within complex applications. In particular, the DL-centered view that underlies the earliest generation of systems, wherein an application was developed in a single environment (the one provided by the DL system), was characterized by very loose interaction, if any, between the DL system and other applications. Later, an approach that viewed the DL more as a component became evident; in this view the DL system acts as a component of a larger environment, typically leaving out functions, such those for data management, that are more effectively implemented by other technologies. The architecture where the component view is taken requires the definition of a clear interface between the components, possibly adopting different modeling languages, but focusing on Description Logics for the implementation of the reasoning services that can add powerful capabilities to the application. Obviously, the choice between the above architectural views depends upon the needs of the application at hand.

Finally, we have already stressed that research in Description Logics has benefited from tight interaction between language designers and developers of DL-KRS. Thus, another major impact on the development of DL research was provided by the implementation of applications using DL-KRS. Indeed, work on DL applications not only demonstrated the effectiveness of Description Logics and of DL-KRS, but also provided mutual feedback within the DL community concerning the weaknesses of both the representation language and the features of an implemented DL-KRS.

1.5.1 Modeling with Description Logics

In order for designers to be able to use Description Logics to model their application domains, it is important for the DL constructs to be easily understandable; this helps facilitate the construction of convenient to use yet effective tools. To this end, the abstract notation that we have previously introduced and that is nowadays commonly used in the DL community is not fully satisfactory.

As already mentioned, there are at least two major alternatives for increasing the usability of Description Logics as a modeling language:

(i) providing a syntax that resembles more closely natural language;
(ii) implementing interfaces where the user can specify the representation structures through graphical operations.

Before addressing the above two possibilities, one brief remark is in order. While alternative ways of specifying knowledge, such as natural language-style syntax, can be more appealing to the user, one should remember that Description Logics in part arose from a need to respond to the inadequacy—the lack of a formal semantic basis—of early semantic networks and frame systems. Those early systems often relied on an assumption of intuitive readings of natural-language-like constructs or graphical structures, which in the end made them unsatisfactory. Therefore, we need to keep in mind always the correspondence of the language used by the user and the abstract DL syntax, and consequently correspondences with the formal semantics should always be clear and available.

The option of a more readable syntax has been pursued in the majority of DL-KRS. In particular, we refer to the concrete syntax proposed in [Patel-Schneider and Swartout, 1993], which is based on a Lisp-like notation, where, for example, the concept of a female person is denoted by `(and Person Female)`. Similarly, the concept $\forall$hasChild.Female would be written `(all hasChild Female)`. In addition, there are shorthand expressions, such as `(the hasChild Female)`, which indicates the existence of a unique female child, and can be phrased using qualified existential restriction and number restriction. In Chapter 10 this kind of syntax is discussed in detail and the possible sources for ambiguities in the natural language reading of the constructs are discussed.

The second option for providing the user with a concrete syntax is to rely on a graphical interface. Starting with the Kl-One system, this possibility has been pursued by introducing a graphical notation for the representation of concepts and roles, as well as their relationships. More recently, Web-based interfaces for Description Logics have been proposed [Welty, 1996a]; in addition, an XML standard has been proposed [Bechhofer *et al.*, 1999; Euzenat, 2001], which is suitable not only for data interchange, but also for providing full-fledged Web interfaces to DL-KRS or applications embodying them as components.

The modeling language is the vehicle for the expression of the modeling notions that are provided to the designers. Modeling in Description Logics requires the designer to specify the concepts of the domain of discourse and characterize their relationships to other concepts and to specific individuals. Concepts can be regarded as classes of individuals and Description Logics as an object-centered modeling language, since they allow one to introduce individuals (objects) and explicitly define

their properties, as well as to express relationships among them. Concept definition, which provides both for necessary and sufficient conditions, is a characteristic feature of Description Logics. The basic relationship between concepts is subsumption, which allows one to capture various kinds of sub-classing mechanisms; however other kinds of relationships can be modeled, such as grouping, materialization, and part-whole aggregation.

The model of a domain in Description Logics is embedded in a knowledge base. We have already addressed the TBox/ABox characterization of the knowledge base. We recall that the roles of TBox and ABox were motivated by the need to distinguish general knowledge about the domain of interest from specific knowledge about individuals characterizing a specific world/situation under consideration. Besides the TBox/ABox, other mechanisms for organizing a knowledge base such as *contexts* and *views* have been introduced in Description Logics. The use of the modeling notions provided by Description Logics and the organization of knowledge bases are addressed in greater detail in Chapter 10.

Finally, we recall that Description Logics as modeling languages overlap to a large extent with other modeling languages developed in fields such as Programming Languages and Database Management. While we shall focus on this relationship later, we recall here that, when compared to modeling languages developed in other fields the characteristic feature of Description Logics is in the reasoning capabilities that are associated with it. In other words, we believe that, while modeling has general significance, the capability of exploiting the description of the model to draw conclusions about the problem at hand is a particular advantage of modeling using Description Logics.

1.5.2 Application domains

Description Logics have been used (and are being used) in the implementation of many systems that demonstrate their practical effectiveness. Some of these systems have found their way into production use, despite the fact that there was no real commercial platform that could be used for developing them.

1.5.2.1 Software engineering

Software Engineering was one of the first application domains for Desciption Logics undertaken at AT&T, where the CLASSIC system was developed. The basic idea was to use a Description Logic to implement a *Software Information System*, i.e., a system that would support the software developer by helping him or her in finding out information about a large software system.

More specifically, it was found that the information of interest for software development was a combination of knowledge about the domain of the application and

code-specific information. However, while the structure of the code can be determined automatically, the connection between code elements and domain concepts needs to be specified by the user.

One of the most novel applications of Description Logics is the LaSSIE system [Devambu *et al.*, 1991], which allowed users to incrementally build a taxonomy of concepts relating domain notions to the code implementing them. The system could thereafter provide useful information in response to user queries concerning the code, such as, for example "the function to generate a dial tone." By exploiting the description of the domain, the information retrieval capabilities of the system went significantly beyond those of the standard tools used for software development. The LaSSIE system had considerable success but ultimately stumbled because of the difficulty of maintenance of the knowledge base, given the constantly changing nature of industrial software. Both the ideas of a Software Information System and the usage of Description Logics survived that particular application and have been subsequently used in other systems. The usage of Description Logics in applications for Software Engineering is described in Chapter 11.

1.5.2.2 Configuration

One very successful domain for knowledge-based applications built using Description Logics is *configuration*, which includes applications that support the design of complex systems created by combining multiple components.

The configuration task amounts to finding a proper set of components that can be suitably connected in order to implement a system that meets a given specification. For example, choosing computer components in order to build a home PC is a relatively simple configuration task. When the number, the type, and the connectivity of the components grow, the configuration task can become rather complex. In particular, computer configuration has been among the application fields of the first Expert Systems and can thus be viewed as a standard application domain for knowledge-based systems. Configuration tasks arise in many industrial domains, such as telecommunications, the automotive industry, building construction, etc.

DL-based knowledge representation systems meet the requirements for the development of configuration applications. In particular, they enable the object-oriented modeling of system components, which combines powerfully with the ability to reason from incomplete specifications and to automatically detect inconsistencies. Using Description Logics one can exploit the ability to classify the components and organize them within a taxonomy. In addition a DL-based approach supports incremental specification and modularity. Applications for configuration tasks require at least two features that were not in the original core of DL-KRS: the representation of rules (together with a rule propagation mechanism), and the ability to provide explanations. However, extensions with so-called "active rules" are now very common

in DL-KRS, and a precise semantic account is given in Chapter 6; significant work on explanation capabilities of DL-KRS has been developed in connection with the design of configuration applications [McGuinness and Borgida, 1995]. Chapter 12 is devoted to the applications developed in Description Logics for configuration tasks.

1.5.2.3 Medicine

Medicine is also a domain where Expert Systems have been developed since the 1980's; however, the complexity of the medical domain calls for a variety of uses for a DL-KRS. In practice, decision support for medical diagnosis is only one of the tasks in need of automation. One focus has been on the construction and maintenance of very large ontologies of medical knowledge, the subject of some large government initiatives. The need to deal with large-scale knowledge bases (hundreds of thousands of concepts) led to the development of specialized systems, such as GALEN [Rector *et al.*, 1993], while the requirement for standardization arising from the need to deal with several sources of information led to the adoption of the DL standard language KRSS [Patel-Schneider and Swartout, 1993] in projects like SNOMED [Spackman *et al.*, 1997].

In order to cope with the scalability of the knowledge base, the DL language adopted in these applications is often limited to a few basic constructs and the knowledge base turns out to be rather shallow, that is to say the taxonomy does not have very many levels of sub-concepts below the top concepts. Nonetheless, there are several language features that would be very useful in the representation of medical knowledge, such as, for example, specific support for PART-OF hierarchies (see Chapter 10), as well as defaults and modalities to capture lack of knowledge (see Chapter 6).

Obviously, since medical applications most often must be used by doctors, a formal logical language is not well-suited; therefore special attention is given to the design of the user interface; in particular, natural language processing (see Chapter 15) is important both in the construction of the ontology and in the operational interfaces.

Further, the DL component of a medical application usually operates within a larger information system, which comprise several sources of information, which need to be integrated in order to provide a coherent view of the available data (on this topic see Chapter 16).

Finally, an important issue that arises in the medical domain is the management of ontologies, which not only requires common tools for project management, such as versioning systems, but also tools to support knowledge acquisition and re-use (on this topic see Chapter 8).

The use of Description Logics specifically in the design of medical applications is addressed in Chapter 13.

1.5.2.4 Digital libraries and Web-based information systems

The relationship between semantic networks and the linked structures implied by hypertext has motivated the development of DL applications for representing bibliographic information and for supporting classification and retrieval in digital libraries [Welty and Jenkins, 2000]. These applications have proven the effectiveness of Description Logics for representing the taxonomies that are commonly used in library classification schemes, and they have shown the advantage of subsumption reasoning for classifying and retrieving information. In these instances, a number of technical questions, mostly related to the use of individuals in the taxonomy, have motivated the use of more expressive Description Logics.

The possibility of viewing the World-Wide Web as a semantic network has been considered since the advent of the Web itself. Even in the early days of the Web, thought was given to the potential benefits of enabling programs to handle not only simple unlabeled navigation structures, but also the information content of Web pages. The goal was to build systems for querying the Web "semantically," allowing the user to pose queries of the Web as if it were a database, roughly speaking. Based on the relationship between Description Logics and semantic networks, a number of proposals were developed that used Description Logics to model Web structures, allowing the exploitation of DL reasoning capabilities in the acquisition and management of information [Kirk *et al.*, 1995; De Rosa *et al.*, 1998].

More recently, there have been significant efforts based on the use of markup languages to capture the information content of Web structures. The relationship between Description Logics and markup languages, such as XML, has been precisely characterized [Calvanese *et al.*, 1999d], thus identifying DL language features for representing XML documents. Moreover, interest in the standardization of knowledge representation mechanisms for enabling knowledge exchange has led to the development of DAML-ONT [McGuinness *et al.*, 2002], an ontology language for the Web inspired by object-oriented and frame-based languages, and OIL [Fensel *et al.*, 2001], with a similar goal of expressing ontologies, but with a closer connection to Description Logics. Since the two initiatives have similar goals and use languages that are somewhat similar (see Chapter 4 for the relationships between frames and Description Logics), their merger is in progress. The use of Description Logics in the design of digital libraries and Web applications is addressed in Chapter 14, with specific discussion on DAML-ONT, OIL, and DAML+OIL.

1.5.2.5 Other application domains

The above list of application domains, while presenting some of the most relevant applications designed with DL-KRS, is far from complete. There are many other domains that have been addressed by the DL community. Among the application

areas that have resorted to Description Logics for useful functions are Planning and Data Mining.

With respect to Planning, many knowledge-based applications rely on the services of a planning component. While Description Logics do not provide such a component themselves, they have been used to implement several general-purpose planning systems. The basic idea is to represent plans and actions, as well as their constituent elements, as concepts. The system can thus maintain a taxonomy of plan types and provide several reasoning services, such as plan recognition, plan subsumption, plan retrieval, and plan refinement. Two examples of planning components developed in a DL-KRS are CLASP [Yen *et al.*, 1991b] developed on top of CLASSIC and EXPECT [Swartout and Gil, 1996], developed on top of LOOM. In addition, the integration of Description Logics and other formalisms, such as Constraint Networks, has been proposed [Weida and Litman, 1992]. Planning systems based on Description Logics have been used in many application domains to support planning services in conjunction with a taxonomic representation of the domain knowledge. Such application domains include, among others, software engineering, medicine, campaign planning, and information integration.

It is worth mentioning that Description Logics have also been used to represent dynamic systems and to automatically generate plans based on such representations. However, in such cases the use of Description Logics is limited to the formalization of properties that characterize the states of the system, while plan generation is achieved through the use of a rule propagation mechanism [De Giacomo *et al.*, 1999]. Such use of Description Logics is inspired by the correspondence between Description Logics and Dynamic Modal Logics described in Chapter 5.

Description Logics have also been used in data mining applications, where their inferences can help the process of analyzing large amounts of data. In this kind of application, DL structures can represent *views*, and DL systems can be used to store and classify such views. The classification mechanism can help in discovering interesting classes of items in the data. We address this type of application briefly in the next subsection on Database Management.

1.5.3 Application areas

From the beginning Description Logics have been considered general purpose languages for knowledge representation and reasoning, and therefore suited for many applications. In particular, they were considered especially effective for those domains where the knowledge could be easily organized along a hierarchical structure, based on the "IS-A" relationship. The ability to represent and reason about taxonomies in Description Logics has motivated their use as a modeling language in the design and maintenance of large, hierarchically structured bodies of knowledge

as well as their adoption as the representation language for formal ontologies [Welty and Guarino, 2001].

We now briefly look at some other research areas that have a more general relationship with Description Logics. Such a relationship exists either because Description Logics are viewed as a basic representation language, as in the case of natural language processing, or because they can be used in a variety of ways in concert with the main technology of the area, as in the field of Database Management.

1.5.3.1 Natural language

Description Logics, as well as semantic networks and frames, originally had natural language processing as a major field for application (see for example [Brachman, 1979]). In particular, when work on Description Logics began, not only was a large part of the DL community working on natural language applications, but Description Logics also bore a strong similarity to other formalisms used in natural language work, such as for example [Nebel and Smolka, 1991].

The use of Description Logics in natural language processing is mainly concerned with the representation of *semantic* knowledge that can be used to convey meanings of sentences. Such knowledge is typically concerned with the meaning of words (the lexicon), and with context, that is, a representation of the situation and domain of discourse.

A significant body of work has been devoted to the problem of disambiguating different syntactic readings of sentences, based on semantic knowledge, a process called *semantic interpretation*. Moreover, semantic knowledge expressed in Description Logics has also been used to support natural language generation.

Since the domain of discourse for a natural language application can be arbitrarily broad, work on natural language has also involved the construction of ontologies [Welty and Guarino, 2001]. In addition, the expressiveness of natural language has led also to investigations concerning extensions of Description Logics, such as for example, default reasoning (see Chapter 6).

Several large projects for natural language processing based on the use of Description Logics have been undertaken, some reaching the level of industrially-deployed applications. They are referenced in Chapter 15, where the role of Description Logics in natural language processing is addressed in more detail.

1.5.3.2 Database management

The relationship between Description Logics and databases is rather strong. In fact, there is often the need to build systems where both a DL-KRS and a DataBase Management System (DBMS) are present. DBMS's deal with persistence of data and with the management of large amounts of it, while a DL-KRS manages intensional knowledge, typically keeping the knowledge base in memory (possibly including as-

sertions about individuals that correspond to data). While some of the applications created with DL-KRS have developed *ad hoc* solutions to the problem of dealing with large amounts of persistent data, in a complex application domain it is very likely that a DL-KRS and a DBMS would both be components of a larger system, and they would work together.

In addition, Description Logics provide a formal framework that has been shown to be rather close to the languages used in semantic data modeling, such as the Entity-Relationship Model [Calvanese *et al.*, 1998g]. Description Logics are equipped with reasoning tools that can bring to the conceptual modeling phase significant advantages, as compared with traditional languages, whose role is limited to modeling. For instance, by using concept consistency one can verify at design time whether an entity can have at least one instance, thus clearly saving all the difficulties arising from discovering such a situation when the database is being populated [Borgida, 1995].

A second dimension of the enhancement of DBMS's with Description Logics involves the query language. By expressing the queries to a database in a Description Logic one gains the ability to classify them and therefore to deal with issues such as query processing and optimization. However, the basic Description Logic machinery needs to be extended in order to deal with conjunctive queries; otherwise DL expressiveness with respect to queries is rather limited. In addition, Description Logics can be used to express constraints and intensional answers to queries.

A corollary of the relationship between Description Logics and DBMS query languages is the utility of Description Logics in reasoning with and about *views*. In the IMACS system [Brachman *et al.*, 1993], the CLASSIC language was used as a "lens" [Brachman, 1994] with which data in a conventional relational database could be viewed. The interface to the data was made significantly more appropriate for a data analyst, and views that were found to be productive could be saved; in fact, they were saved in a taxonomy and could be classified with respect to one another. In a sense, this allows the schema to be viewed and queried explicitly, something normally not available when using a raw DBMS directly.

A more recent use of Description Logics is concerned with so-called "semi-structured" data models [Calvanese *et al.*, 1998c], which are being proposed in order to overcome the difficulties in treating data that are not structured in a relational form, such as data on the Web, data in spreadsheets, etc. In this area Description Logics are sufficiently expressive to represent models and languages that are being used in practice, and they can offer significant advantages over other approaches because of the reasoning services they provide.

Another problem that has recently increased the applicability of Description Logics is information integration. As already remarked, data are nowadays available in large quantities and from a variety of sources. Information integration is the task

of providing a unique coherent view of the data stored in the sources available. In order to create such a view, a proper relationship needs to be established between the data in the sources and the unified view of the data. Description Logics not only have the expressiveness needed in order to model the data in the sources, but their reasoning services can help in the selection of the sources that are relevant for a query of interest, as well as to specify the extraction process [Calvanese *et al.*, 2001c].

The uses of Description Logics with databases are addressed in more detail in Chapter 16.

1.6 Extensions of Description Logics

In this section we look at several types of extensions that have been proposed for Description Logics; these are addressed in more detail in Chapter 6. Such extensions are generally motivated by needs arising in applications. Unfortunately, some extended features in implemented DL-KRS were created without precise, formal accounts; in some other cases, such accounts have been provided using a formal framework that is not restricted to first-order logic.

A first group of extensions has the purpose of adding to DL languages some representational features that were common in frame systems or that are relevant for certain classes of applications. Such extensions provide a representation of some novel epistemological notions and address the reasoning problems that arise in the extended framework.

Extensions of a second sort are concerned with reasoning services that are useful in the development of knowledge bases but are typically not provided by DL-KRS. The implementation of such services relies on additional inference techniques that are considered non-standard, because they go beyond the basic reasoning services provided by DL-KRS.

Below we first address the extensions of the knowledge representation framework and then non-standard inferences.

1.6.1 Language extensions

Some of the research associated with language extensions has investigated the semantics of the proposed extensions, but often the emphasis is only on finding reasoning procedures for the extended languages. Within these language extensions we find constructs for non-monotonic, epistemic, and temporal reasoning, and constructs for representing belief and uncertain and vague knowledge. In addition some constructs address reasoning in concrete domains.

1.6.1.1 Non-monotonic reasoning

When frame-based systems began to be formally characterized as fragments of first-order logic, it became clear that those frame-based systems as well as some DL-KRS that were used in practice occasionally provided the user with constructs that could not be given a precise semantic characterization within the framework of first-order logic. Notable among the problematic constructs were those associated with the notion of defaults, which over time have been extensively studied in the field of non-monotonic reasoning [Brachman, 1985].

While one of the problems arising in semantic networks was the oft-cited so-called "Nixon diamond" [Reiter and Criscuolo, 1981], a whole line of research in non-monotonic reasoning was developed in trying to characterize the system behavior by studying structural properties of networks. For example, the general property that "birds fly" might not be inherited by a penguin, because a rule that penguins do not fly would give rise to an arc in the network that would block the default inference. But as soon as the network becomes relatively complex (see for example [Touretzky *et al.*, 1991]), we can see that attempts to provide semantic characterization in terms of network structure are inadequate.

Another approach that has been pursued in the formalization of non-monotonic reasoning in semantic networks is based on the use of default logic [Reiter, 1980; Etherington, 1987; Nado and Fikes, 1987]. Following a similar approach is the treatment of defaults in DL-based systems [Baader and Hollunder, 1995a], where formal tools borrowed from work on non-monotonic reasoning have been adapted to the framework of Description Logics. Such adaptation is non-trivial, however, because Description Logics are not, in general, propositional languages.

1.6.1.2 Modal representation of knowledge and belief

Modal logics have been widely studied to model a variety of features that in first-order logic would require the application of special constraints on certain elements of the formalization. For example, the notions of knowing something or believing that some sentence is true can be captured by introducing modal operators, which characterize properties that sentences have.

For instance the assertion

$$\mathbf{B}(\mathsf{Married}(\mathsf{ANNA}))$$

states a fact explicitly concerning the system's beliefs (the system believes that Anna is married), rather than asserting the truth of something about the world being modeled (the system could believe something to be true without firm knowledge about its truth in the world).

In general, by introducing a modal operator one gains the ability to model properties like knowledge, belief, time-dependence, obligation, and so on. On the one

hand, extensions of Description Logics with modal operators can be viewed very much like the corresponding modal extensions of first-order logic. In particular, the semantic issues arising in the interpretation of quantified modal sentences (i.e., sentences with modal operators appearing inside the scope of quantifiers) are the same. On the other hand, the syntactic restrictions that are suited to a DL language lead to formalisms whose expressiveness and reasoning problems inherit some of the features of a specialized DL language. Extensions of Description Logics with modal operators including those for representing knowledge and belief are discussed in [Baader and Ohlbach, 1995].

1.6.1.3 Epistemic reasoning

It is not sufficient to provide a semantics for defaults to obtain a full semantic account of frame-based systems. Frame-based systems have included procedural rules as well as other forms of closure and epistemic reasoning that need to be covered by the semantics as well as by the reasoning algorithms. In particular, if one looks at the most widely-used systems based on Description Logics, such features are still present, possibly in new flavors, while their semantics is given informally and the consequences of reasoning sometimes not adequately explained.

Among the non-first-order features that are used in the practice of knowledge-based applications in both DL-based and frame-based systems we point out these:

- *procedural rules*, (also called *trigger rules*) which are normally described as *if-then* statements and are used to infer new facts about known individuals;
- *default rules*, which enable default reasoning in inheritance hierarchies;
- *role closure*, which limits the reasoning involving role restrictions to the individuals explicitly in the knowledge base;
- *integrity constraints*, which provide consistency restrictions on admissible knowledge bases.

In Chapter 6, among other approaches an epistemic extension of Description Logics with a modal operator is addressed. In the resulting formalism [Donini *et al.*, 1998a] one can express epistemic queries and, by admitting a simple form of epistemic sentences in the knowledge base, one can formalize the aforementioned procedural rules. This characterization of procedural rules in terms of an epistemic operator has been widely accepted in the DL community and is thus also included in Chapter 2. The approach has been further extended to what have been called Autoepistemic Description Logics (ADLs) [Donini *et al.*, 1997b], where it is combined with default reasoning. This combination is achieved by relying on the non-monotonic modal logic *MKNF* [Lifschitz, 1991], thus introducing a second modal operator interpreted as autoepistemic assumption. The features mentioned above can be uniformly treated as epistemic sentences in the knowledge base, without the

need to give them special status as in the case of procedural rules, defaults, and epistemic constraints on the knowledge base. This expressiveness does not come without making reasoning more difficult. An extension of the reasoning methods available for deduction in the propositional formalizations of non-monotonic reasoning to the fragment of first-order logic corresponding to Description Logics has nonetheless been shown to be decidable.

1.6.1.4 Temporal reasoning

One notion that is often required in the formalization of application domains is time. Temporal extensions of Description Logics have been treated as a special kind of modal extension. The first proposal for handling time in a DL framework [Schmiedel, 1990] was originated in the context of the DL system BACK. Later, following the standard approaches in the representation of time, both interval-based and point-based approaches have been studied, specifically focusing on the decidability and complexity of the reasoning problems (see [Artale and Franconi, 2001] for a survey the temporal extensions of Description Logics).

Time intervals can also be treated as a form of concrete domain (see below).

1.6.1.5 Representation of uncertain and vague knowledge

Another aspect of knowledge that is sometimes useful in representing and reasoning about application domains is uncertainty. As in other knowledge representation frameworks there are several approaches to the representation of uncertain knowledge in Description Logics. Two of them, namely probabilistic logic and fuzzy logic, have been proposed in the context of Description Logics. In the case of probabilistic Description Logics [Heinsohn, 1994; Jaeger, 1994] the knowledge about the domain is expressed in terms of probabilistic terminological axioms, which allow one to represent statistical information about the domain, and in terms of probabilistic assertions, which specify the degree of belief of asserted properties. The reasoning tasks aim at finding the probability bounds for subsumption relations and assertions. A more recent line of work tries to combine Description Logics with Bayesian networks.

In the case of fuzzy Description Logics [Yen, 1991] the goal is to characterize notions that cannot be properly defined with a "crisp" numerical bound. For example, the concept of living near Rome cannot be always defined with a crisp boundary on the map, but must be represented with a membership or degree function, which expresses closeness to the city in a continuous way.

Proposed approaches to fuzzy Description Logics not only define the semantics of assertions in terms of fuzzy sets, but also introduce new operators to express notions like "mostly," "very," etc. Reasoning algorithms are also provided for computing fuzzy subsumption within the framework of tableau-based methods.

1.6.1.6 Concrete domains

One of the limitations of basic Description Logics is related to the difficulty of integrating knowledge (and, consequently, performing reasoning) of specific domains, such as numbers or strings, which are needed in many applications. For example, in order to model the concept of a young person it seems rather natural to introduce the (functional) role *age* and to use a concrete value (or range of values) in the definition of the concept. In addition, one would like to be able to conclude that a person of school age is also a young person. Such a conclusion might require the use of properties of numbers to establish that the expected subsumption relation holds.

While for some time such extensions were designed in *ad hoc* ways, in [Baader and Hanschke, 1991a] a general method was established for integrating knowledge about concrete domains within a DL language. If a domain can be properly formalized, it is shown that the tableau-based reasoning technique can be suitably extended to handle the reasoning services in the extended language.

Concrete domains include not only data types such as numerical types, but also more elaborate domains, such as tuples of the relational calculus, spatial regions, or time intervals.

1.6.2 Additional reasoning services

Non-standard inference tasks can serve a variety of purposes, among them support in building and maintaining the knowledge base, as well as in obtaining information about the knowledge represented in it.

Among the more useful non-standard inference tasks in Description Logics we find the computation of the least common subsumer and the most specific concept, matching/unification, and concept rewriting.

1.6.2.1 Least common subsumer and most specific concept

The least common subsumer (*lcs*) of a set of concepts is the minimal concept that subsumes all of them. The minimality condition implies there is no other concept that subsumes all the concepts in the set and is less general (subsumed by) the *lcs*. This notion was first studied in [Cohen *et al.*, 1992] and it has subsequently been used for several tasks: inductive learning of concept description from examples; knowledge base vivification (as a way to represent disjunction in languages that do not admit it); and in the bottom-up construction of DL knowledge bases (starting from instances of the concepts).

The notion of *lcs* is closely related to that of most specific concept (*msc*) of an individual, i.e., the least concept description that the individual is an instance of, given the assertions in the knowledge base; the minimality condition is specified

as before. More generally, one can define the *msc* of a set of assertions about individuals as the *lcs* of the *msc* associated with each individual. Based on the computation of the *msc* of a set of assertions about individuals one can incrementally construct a knowledge base [Baader and Küsters, 1999].

It interesting to observe that the techniques that have been proposed to compute the *lcs* and *mcs* rely on compact representations of concept expressions, which are built either following the structural subsumption approach, or through the definition of a well-suited normal form.

1.6.2.2 Unification and matching

Another tool to support the construction and maintenance of DL knowledge bases that goes beyond the standard inference services provided by DL-KRS is the unification of concepts.

Concept unification [Baader and Narendran, 1998] is an operation that can be regarded as weakening the equivalence between two concept expressions. More precisely, two concept expressions unify if one can find a substitution of concept variables into concept expressions such that the result of applying the substitution gives equivalent concepts. The intuition is that, in order to find possible overlaps between concept definitions, one can treat certain concept names as variables and discover, via unification, that two concepts (possibly independently defined by distinct knowledge designers) are in fact equivalent. The knowledge base can consequently be simplified by introducing a single definition of the unifiable concepts.

As usual, matching is defined as a special case of unification, where variables occur only in one of the two concept expressions. In addition, in the framework of Description Logics, one can define matching and unification based on the subsumption relation instead of equivalence [Baader *et al.*, 1999a].

As with other non-standard inferences, the computation of matching and unification relies on the use of specialized representations for concept expressions, and it has been shown to be decidable for rather simple Description Logics.

1.6.2.3 Concept rewriting

Finally, there has been a significant body of work on the problem of Concept Rewriting. Given a concept expressed in a source language, Concept Rewriting amounts to finding a concept, possibly expressed in a target language, which is related to the given concept according to equivalence, subsumption, or some other relation.

In order to specify the rewriting, one can provide a suitable set of constraints between concepts in the source language and concepts in the target language. Concept Rewriting can be applied to the translation of concepts from one knowledge base to another, or in the reformulation of concepts during the process of knowledge base construction and maintenance.

In addition, Concept Rewriting has been addressed in the context of the rewriting of queries using views, in Database Management (see also Chapter 16), and has recently been investigated in the framework of Information Integration. In this setting, one can apply Concept Rewriting techniques to automatically generate the queries that enable a system to gather information from a set of sources [Beeri *et al.*, 1997]. Given an initial specification of the query according to a common, global language, and a set of constraints expressing the relationship between the global schema and the individual sources where information is stored, the problem is to compute the queries to be posed to the local sources that provide answers, possibly approximate, to the original query [Calvanese *et al.*, 2000a].

1.7 Relationship to other fields of Computer Science

Description Logics were developed with the goals of providing formal, declarative meanings to semantic networks and frames, and of showing that such representation structures can be equipped with efficient reasoning tools. However, the underlying ideas of concept/class and hierarchical structure based upon the generality and specificity of a set of classes have appeared in many other field of Computer Science, such as Database Management and Programming Languages. Consequently, there have been a number of attempts to find commonalities and differences among formalisms with similar underlying notions, but which were developed in different fields. Moreover, by looking at the syntactic form of Description Logics—logics that are restricted to unary and binary predicates and allow for restricted forms of quantification—other, logical formalisms that have strong relationships with Description Logics have been identified. In this section we briefly address such relationships; in particular, we focus our attention on the relationship of Description Logics to other class-based languages, and then we address the relationship between Description Logics and other logics. These topics are addressed in more detail in Chapter 4.

1.7.1 Description Logics and other class-based formalisms

As we have mentioned, Description Logics can, in principle, be related to other class-based formalisms. Before looking at other fields, it is worth relating Description Logics to other formalisms developed within the field of Knowledge Representation that share the intuitions underlying network-based representation structure. In [Lehmann, 1992] several languages aiming at structured representations of knowledge are reviewed. We have already discussed the relationship between Description Logics and semantic networks and frames, since they provided the basic motivations for developing Description Logics in the first place. Among others, *conceptual*

graphs [Sowa, 1991] have been regarded as a way of representing conceptual structures very closely related to semantic networks (and consequently, to Description Logics). However, only recently has there been a detailed analysis of the relationship between conceptual graphs and Description Logics[Baader *et al.*, 1999c]. The outcome of this work makes it apparent that, although one can establish a relationship between simple conceptual graphs and a DL language, there are substantial differences between the two formalisms. The most significant one is that Description Logics are characterized by the universally quantified role restriction, which is not present in conceptual graphs. Consequently, the interpretation of the representation structures becomes substantially different.

In many other fields of Computer Science we find formalisms for the representation of objects and classes [Motschnig-Pitrik and Mylopoulous, 1992]. Such formalisms share the notion of a class that denotes a subset of the domain of discourse, and they allow one to express several kinds of relationships and constraints (e.g., subclass constraints) that hold among classes. Moreover, class-based formalisms aim at taking advantage of the class structure in order to provide various types of information, such as whether an element belongs to a class, whether a class is a subclass of another class, and more generally, whether a given constraint holds between two classes. In particular, formalisms that are built upon the notions of class and class-based hierarchies have been developed in the field of Database Management, in semantic data modeling (see for example [Hull and King, 1987]), in object-oriented languages (see for example [Kim and Lochovsky, 1989]), and more generally, in Programming Languages (see for example [Lenzerini *et al.*, 1991]).

There have been several attempts to establish relationships among the class-based formalisms developed in different fields. In particular, the common intuitions behind classes and concepts have stimulated several pieces of work aimed at establishing a precise relationship between class-based formalisms and Description Logics. However, it is difficult to find a common framework for carrying out a precise comparison.

In Chapter 4 a specific Description Logic is taken as a basis for identifying the common features of frame systems and object-oriented and semantic data models (see also [Calvanese *et al.*, 1999e]). Specifically, a precise correspondence between the chosen DL and the Entity-Relationship model [Chen, 1976], as well as with an object-oriented language in the style of [Abiteboul and Kanellakis, 1989], is presented there.

This kind of comparison shows that one can indeed identify a large common basis, but also that there are features that are currently missing in each formalism. For example, to capture semantic data models one needs a cyclic form of inclusion assertion, as well as the *inverses* of roles for modeling relationships that work in both directions, while DL roles have a directionality from one concept to another.

Moreover, in order to make a comparison with frame-based systems, one has to leave out both the non-monotonic features of frames, such as defaults and closures (that are addressed among the extensions of Description Logics in the previous section) and their dynamic aspects such as daemons and and triggers (with the exception of trigger rules, which are also addressed in the previous section). Finally, with respect to object-oriented data models the main difference is that although Description Logics provide the expressiveness to model record and set structures, they are not explicitly available in Description Logics and thus their representation is a little cumbersome. On the other hand, semantic and object-oriented data models are typically not equipped with reasoning tools that are available with Description Logics. This issue is further developed in Chapter 16, where the applications of Description Logics in the field of Database Management are addressed. However, if the language is sufficiently expressive, as it needs to be in order to establish relationships among various class-based formalisms, one needs to distinguish between *finite model* reasoning which is required for Database languages that are designed to represent a closed domain of discourse, and *unrestricted* reasoning, which is typical of knowledge representation formalisms and, therefore, of Description Logics.

1.7.2 Relationships to other logics

The initial observation for addressing the relationship of Description Logics to other logics is the fact that Description Logics are subsets of first-order logic. This has been known since the earliest days of Description Logics, and has been thoroughly investigated in [Borgida, 1996]. In fact, the DL $\mathcal{ALC}$ corresponds to the fragment of first-order logic obtained by restricting the syntax to formulas containing two variables. The importance of this and subsequent studies on this issue is related to finding adequate characterizations of the expressiveness of Description Logics.

Since Description Logics focus on a language formed by unary and binary predicates, it turned out that they are closely related to modal languages, if one regards roles as accessibility relations. In particular, Schild [1991] pointed out that some Description Logics are notational variants of certain propositional modal logics; specifically, the DL $\mathcal{ALC}$ has a modal logic counterpart, namely the multi-modal version of the logic K (see [Halpern and Moses, 1992]). Actually, $\mathcal{ALC}$-concepts and formulas in multi-modal **K** can immediately be translated into each other. Moreover, an $\mathcal{ALC}$-concept is satisfiable if and only if the corresponding **K**-formula is satisfiable. Research in the complexity of the satisfiability problem for modal propositional logics was initiated quite some time before the complexity of Description Logics was investigated. Consequently, this relationship made it possible to borrow from modal logic complexity results, reasoning techniques, and language constructs that had not been previously considered in Description Logics. On the

other hand, there are features of Description Logics that did not have counterparts in modal logics and therefore needed *ad hoc* extensions of the reasoning techniques developed for modal logics. In particular, number restrictions as well as the treatment of individuals in the ABox required specific treatments based on the idea of *reification*, which amounts to expressing the extensions through a special kind of axiom within the logic. Finally, we mention that recent work has pointed out a relationship between Description Logics and guarded fragments, which can be regarded as generalizations of modal logics. Most of the research on very expressive Description Logics, addressed in Chapter 5, has its roots in the correspondence with modal logic.

1.8 Conclusion

From their humble origins in the late 1970's as a remedy for logical and semantic problems in frame and semantic network representations, Description Logics have grown to be a unique and important keystone in the history of Knowledge Representation. DL formalisms certainly evoked interest in their earliest days, with the invention and application of the KL-ONE system, but international attention and research was given a significant boost in 1984 when Brachman and Levesque used the simple and intuitive structure of Description Logics as the basis for their observation about the tradeoff between knowledge representation language expressiveness and computational complexity of reasoning. The way Description Logics were able to separate out the structure of concepts and roles into simple term-forming operators opened the door to extensive analysis of a broad family of languages. One could add and subtract these operators from the language and explore both the computational ramifications and the relationship of the resulting language to other formal languages in Computer Science, such as modal logics and data models for database systems.

As a result, the family of Description Logic languages is probably the most thoroughly understood set of formalisms in all of knowledge representation. The computational space has been thoroughly mapped out, and a wide variety of systems have been built, testing out different styles of inference computation and being used in many applications.

Description Logics are responsible for many of the cornerstone notions used in knowledge representation and reasoning. They helped crystallize many of the ideas treated informally in earlier notations, such as concepts and roles. But they added many new important building blocks for later work in the field: the terminology/assertion distinction (TBox/ABox), number and value restrictions on roles, internal structure for concepts, Tell/Ask interfaces, and others. They have been the subject of a great deal of comparison and analysis with their cousins in other fields

of Computer Science, and DL systems run the gamut from simple, restricted systems with provably advantageous computational properties to extremely expressive systems that can support very powerful applications. Perhaps, the most important aspect of work on Description Logics has been the very tight coupling between theory and practice. The exemplary give-and-take between the formal, analytical side of the field and the pragmatic, implemented side—notable throughout the entire history of Description Logics—has been a role model for other areas of AI.

Acknowledgements

We are grateful to Franz Baader, Francesco M. Donini, Maurizio Lenzerini, and Riccardo Rosati for reading the manuscript and making suggestions for improving the final version of the chapter.