1

An Introduction to Description Logics

Daniele Nardi
Ronald J. Brachman

Abstract

This introduction presents the main motivations for the development of Description Logics (DLs) 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 DL languages and related reasoning techniques.

DL 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 ability 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 1970s – 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 to 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 on 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" 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 domains (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*. 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 or 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 woman 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

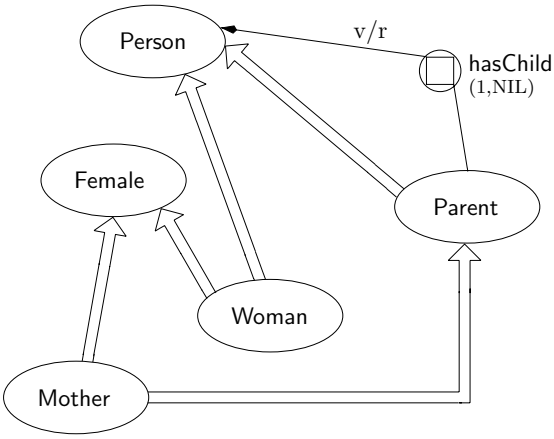

Fig. 1.1. An example network.

[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 "individuals 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 role fillers. 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, hasChild $\sqcap$ 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.