Description Logics in Data Management
Alexander Borgida

Abstract—Description logics and reasoners, which are descendants of the KL-ONE language, have been studied in depth in Artificial Intelligence. After a brief introduction, we survey in this paper their application to the problems of information management, using the framework of an abstract information server equipped with several operations—each involving one or more languages. Specifically, we indicate how one can achieve enhanced access to data and knowledge by using descriptions in languages for schema design and integration, queries, answers, updates, rules, and constraints.

Index Terms—Description, concept, terminological, language, subsumption, knowledge representation, object-oriented schema, intension.

I. INTRODUCTION

A large class of practical computer applications requires managing a symbolic model of an application world, which is updated or queried by users. Many such systems start with the intuition that for describing some situation, it is useful to think of various kinds of individuals, e.g., Calculus100, Gauss, related by relationships, e.g., taughtBy, and grouped into classes, e.g., COURSE, TEACHER, STUDENT. This intuition is shared by formalisms such as semantic data models, object-oriented databases, and semantic networks. Such formalisms support languages for declaring classes of individuals, using a syntax somewhat resembling the following example:

class ADVANCED_COURSE is-a COURSE with
takers [0,40]: GRADS

Such a declaration is intended to express necessary conditions that must be met by each instance of the class. For example, in the above case every instance of ADVANCED_COURSE must also be an instance of class COURSE, and the takers attribute must relate to it between 0 and 40 individuals, themselves instances of class GRADS. Class definitions are used to detect errors, or as a template for data storage decisions, i.e., as a type declaration in standard programming languages.

The subject of this paper is yet another family of formalisms—description logics (DLs)—which are currently enjoying a surge of interest both as objects of theoretical study and as tools used in applications, including ones in industry.

A. Description Languages

The fundamental observation underlying DLs is that there is a benefit to be gained if languages for talking about classes of individuals yield structured objects that can be reasoned with.

Fig. 1 contains an example of a typical compositional description, expressed in the CLASSIC language [17]. Its intended reading would be "Courses with at most 10 takers, all takers being instances of GRADS”. In this description, COURSE and GRADS are identifiers for concepts introduced elsewhere, while takers is the name of a binary relation, intended to relate courses to students taking them. There are several things one can do with such a description, including:

- **Reasoning** about the relationship of one description to another, treating them as “intensional” objects. For example, the description in Fig. 1 is subsumed by (entails) the description (and COURSE, (at-most 15 takers)) since everything with at most 10 fillers for some role, also has at most 15 fillers for it. On the other hand, the description (at-least 12 takers) can be inferred to be disjoint from the one in Fig. 1, because the required number of fillers are in conflict.
- **Recognizing** those individuals that satisfy the description, based on what is currently known about them. For example, suppose A100 is an individual object in the knowledge base, and it is known to be an instance of the concept COURSE; in addition, the fillers for the takers role for A100 are individuals Calvin and Hobbes, both of which are instances of GRADS. Then A100 is inferred to be an instance of the description in Fig. 1, since all the necessary and sufficient conditions of that concept are satisfied.

```
and

COURSE
(at-most 10 takers),
(all takers GRADS)
```

Fig. 1. Compositional concept in CLASSIC.

As a possible clarification of the issues involved, we provide an analogy for those familiar with logic programming.

Since descriptions denote concepts or relationships, it is natural to take their analogues in logic to be ordinary unary or binary predicates. Consider the following knowledge base of Horn clauses:

- `ParentOf(liz, andy).`
- `Male(andy).`
- `Child(_x) :- ParentOf(_z, _x).`
- `Son(_y) :- Male(_y), ParentOf(_w, _y).`

Normally, such a system is used to deduce new properties of individuals, e.g., whether the Son predicate "recognizes" the individual andy. On the other hand, we might want to reason entirely from intensional information—the rules—ignoring ground facts. For example, we might be interested in whether Child(_x) is implied by ("subsumes") Son(_x). Note that

Manuscript received July 18, 1994.
The author is with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA. E-mail: borgida@cs.rutgers.edu.
IEEECS Log Number K95057.
although we cannot express this question in Prolog, theoretically the answer would be "yes," because the last two clauses are in fact treated as the following definitions

\[ \text{Child}(x) \iff (\exists x) \text{ParentOf}(z, x) \]
\[ \text{Son}(y) \iff (\exists w) \text{Male}(y) \land \text{ParentOf}(w, y) \]

by the semantics of predicate completion in Prolog. However, if we took seriously the rule for Son as its definition, then asserting Son(fred) ought to allow us to deduce that Child(fred)—a deduction not made in current logic programming systems. It is such reasoning with definitions that is the trademark of description logics.

B. Outline

For readers familiar with database management, the paper provides a tutorial and survey of how descriptions and their reasoners can enhance the modeling power of the database (i.e., the kinds of knowledge about the world that can be stored), facilitate the user’s interaction with it, or support the development of databases.

For readers conversant with Artificial Intelligence, DLs are descendants of the influential KL-ONE system [20], [21], and have been extensively studied under the name of “terminological logics.” The features and history of these logics have been surveyed recently in papers such as [66], [46]. Therefore our aim is to provide for this audience a novel, systematic look at the various uses to which DLs are being put for information management—a view considerably broader than that usually assumed in Artificial Intelligence.

We begin by considering the syntax and semantics of description languages, illustrating the kinds of reasoning they are especially suited for. Thereafter, we present a semi-formal view of an Information System as a “black box” with several operations, each of them involving one or more languages. By examining the possibility of using DLs for each of these languages, we obtain a systematic survey of their utility.

Throughout the paper we endeavor to summarize the key points as italicized observations.

II. THE SYNTAX AND SEMANTICS OF DLs

Although the original KL-ONE system supported a graphical notation for representing definitions of concepts, all DLs since the KRYPTON system [23] provide a formal linear syntax for writing descriptions. To give the reader a sense of the syntactic variations in use, here are versions of the description in Fig. 1 in the two other currently most widely used DLs, BACK [44] and LOOM [45], as well as an infix notation used in many theoretical papers:

\[ \text{and} \left( \text{COURSE}, \right. \]
\[ \text{at-most} \left( 20, \text{takers}, \right. \]
\[ \text{all} \left( \text{taughtBy, \text{prim} \left( \text{PROFESSOR} \right)} \right) \]

is intended to denote those instances of the (primitive) concept COURSE which have at most 20 students taking them (at most 20 fillers for the takers role), and are only taught by PROFESSORS. Suppose professors have a rank role, whose possible values are in the set \{AP, AP, P\}, and suppose we wish to restrict the above description to include only courses taught by tenured professor—i.e., add a restriction that the composition of taughtBy with rank must be one of the values AP or P. This can be accomplished by adding an additional conjunct, built using concept constructor one-of, which takes as arguments an enumeration of values, and role constructor compose, which denotes role composition:

\[ \text{all} \left( \text{compose} \left( \text{taughtBy, rank} \right), \text{one-of} \left( \text{AP, P} \right) \right) \]

Table I contains a fairly comprehensive list of domain-independent description constructors, from [66], which were arrived at empirically, in efforts to express the meaning of natural language sentences and other Artificial Intelligence tasks.

One can in fact view DLs as a logical notation where logical operators were chosen to facilitate the expression of frequently used conceptual structures, and related inferences. To highlight this point, consider the alternative of representing descriptions as unary and binary predicates in Predicate Calculus. The formula, with free variable \( \alpha \), corresponding to the description example in this section is

\[ \text{COURSE}(\alpha) \]
\[ \land ( \exists x_1 \ldots \exists x_{20} ) \text{takers}(\alpha, x_1) \land \ldots \land \text{takers}(\alpha, x_{20}) \]
\[ \land ( x_1 \neq x_2 \land x_1 \neq x_3 \land \ldots \land x_{19} \neq x_{20} ) \]
\[ \land \forall r \text{taughtBy}(\alpha, r) \supset \text{PROFESSOR}(r) \]
\[ \land \forall r \forall y \text{taughtBy}(\alpha, r) \land \text{rank}(r, y) \supset \]
\[ ( y = \text{AP} \lor y = \text{P} ) \]

It is evident that the encoding in predicate logic is less perspicuous, mostly due to the proliferation of variables and quantifiers. As a result, it is more difficult to represent information in this notation, and it is less readable for humans. It is also more difficult for theorem provers to recognize the subsets of the above sentences which are amenable to fast but special purpose reasoning, e.g., checking that at-least(25, takers) entails at-least(20, takers) is a matter of a single integer comparison for DL-based reasoners. An interesting distin-

2. Such readers are assumed to have elementary familiarity with propositional and first order logic.
3. For this purpose, some elementary familiarity with the functionality of relational databases is assumed.
giving feature of the syntax of description languages is that they express such statements without introducing the notion of variable, scoping, and substitution.

We summarize the preceding in the following observation:

**DLs provide languages for building variable-free, composite terms from primitive identifiers using term constructors; these terms denote several sorts of things, including concepts (sets of individuals) and roles (relationships, which are usually binary).**

### Table 1

**Domain-independent description constructors**

<table>
<thead>
<tr>
<th>TERM</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>top-concept</td>
<td>$\Delta^2$</td>
</tr>
<tr>
<td>nothing</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>and(C,D)</td>
<td>$C^2 \cap D^2$</td>
</tr>
<tr>
<td>or(C,D)</td>
<td>$C^2 \cup D^2$</td>
</tr>
<tr>
<td>not(C)</td>
<td>$\Delta^2 \setminus C^2$</td>
</tr>
<tr>
<td>all[p,C]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>some[p,C]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>at-least[n,p]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>at-most[n,p]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>at-least-n-c(p,C)</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>at-most-n-c(p,C)</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>same-as[p,q]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>subset[p,q]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>not-same-as[p,q]</td>
<td>${d \in \Delta^2</td>
</tr>
<tr>
<td>fill[p,b]</td>
<td>${d \in \text{dom}</td>
</tr>
<tr>
<td>not-fill[p,b]</td>
<td>${d \in \text{dom}</td>
</tr>
<tr>
<td>one-of[b1,...,bn]</td>
<td>${b^1, ..., b^n}$</td>
</tr>
</tbody>
</table>

### Table 2

**Domain-independent role constructors**

<table>
<thead>
<tr>
<th>TERM</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>top-role</td>
<td>$\Delta^2 \times \Delta^2$</td>
</tr>
<tr>
<td>identity</td>
<td>${(d,d)</td>
</tr>
<tr>
<td>role-and[p,q]</td>
<td>$p^2 \cap q^2$</td>
</tr>
<tr>
<td>role-or[p,q]</td>
<td>$p^2 \cup q^2$</td>
</tr>
<tr>
<td>role-not[p]</td>
<td>$\Delta^2 \times \Delta^2 \setminus \Delta^2$</td>
</tr>
<tr>
<td>inverse[p]</td>
<td>${(d,d')</td>
</tr>
<tr>
<td>restrict[p,C]</td>
<td>${(d,d')</td>
</tr>
<tr>
<td>compose[p,q]</td>
<td>$p^2 \circ q^2$</td>
</tr>
<tr>
<td>product[C,D]</td>
<td>$C^2 \times D^2$</td>
</tr>
<tr>
<td>trans[p]</td>
<td>$\bigcup_{n&gt;0} (p^n)^2$</td>
</tr>
</tbody>
</table>

### A. The Logic of Descriptions

We have seen above that an interpretation associates with every concept description an extent, just like the interpretation of a unary predicate in FOPC. There are a number of natural questions that one normally asks about a description $D$

- **Is $D$ coherent/consistent?**: The answer is no if the denotation of $D$, $D'$, is empty for every possible relational structure $I$.
- **Does $D$ subsume $C$?**: The answer is yes if the denotation of $C$ is a subset of the denotation of $D$, $C' \subseteq D'$, for every possible relational structure $I$.
- **Are $D$ and $C$ mutually disjoint?**: The answer is yes if $C' \cap D' = \emptyset$ for every possible relational structure $I$.
- **Are $D$ and $C$ equivalent?**: The answer is yes if $C' = D'$ for every possible relational structure $I$.

The subsumption relationship, which corresponds to material implication between predicates and is symbolized by $\Rightarrow$, is usually considered the most basic one. This is because all DLs have concept constructors and and nothing (which denotes the inconsistent concept with empty extension), so that incoherence can be detected by asking the question "$D \Rightarrow $ nothing?", while disjointness is answered by "and($C,D) \Rightarrow $ nothing?", and equivalence ($\equiv$) is mutual subsumption.

In fact, the presence of and and nothing allows us to view the space of all descriptions (actually, this space modulo the $\equiv$ relation), partially ordered by $\Rightarrow$, to be a mathematical structure called a "meet-semilattice", where every pair of descriptions $B$ and $C$ has a greatest lower bound—a description that subsumes any other description that is subsumed by both $B$ and $C$—namely, and($B,C$). We emphasize that it is the entire infinite space of descriptions that is the semi-lattice, not just some finite subset of named descriptions of interest for some particular application or other.

Although not the norm ([1], [29], [64] are exceptions), we believe that it is important to pursue this lattice-theoretic approach as foundational for DLs, if we wish to treat concepts as structured, intensional objects. Actually, the DLs encountered in practice usually form full lattices: there is a most general concept, top-concept, and there exists a unique least common subsumer for every pair of descriptions. Moreover, given descriptions $B$ and $C$ such that $B \Rightarrow C$, it is possible to consider the notion of relative complement(s) in this lattice: the maximal description(s) $D$ such and($C,D) = B$.

For example, Fig. 2 presents two descriptions, their greatest common subsumer (meet), least common subsumer (join), and the relative complement between the join in (c) and the first concept in (a), in a language having constructors {and, at-most, at-least, all-one-of}.

**For example, Figure 2 presents two descriptions, their greatest common subsumer (meet), least common subsumer (join), and the relative complement between the join in (c) and the first concept in (a), in a language having constructors {and, at-most, at-least, all-one-of}.**

### (a) Two descriptions.

```latex
\text{and(\text{prin(COURSE),})}
\text{at-most(25, takers),}
\text{all(\text{taughtBy, one-of(Gauss, Euclid))})}
```

### (b) The meet of the two descriptions in part (a)

```latex
\text{and(\text{prin(COURSE), prin(FUNNY-EVENT)})}
\text{at-most(20, takers),}
\text{all(\text{taughtBy, one-of(Gauss, Marx))})}
```

### (c) The join of the two descriptions in part (a)

```latex
\text{all(\text{taughtBy, one-of(Gauss, Euclid, Marx))})}
```

### (d) The relative complement of the description in (c) and the first description in (a).

```
```

Fig. 2. Two descriptions and lattice operations on them.

The interested reader may consult [29], [64] for conditions
under which the semi-lattice is guaranteed to be a lattice and to
have unique relative complements.

The domain of concept terms of a DL together with the sub-
superset relation ⊆, and the and constructor (or its equiva-
 lent) form a semi-lattice, in which it is usually possible to also
define join and relative complement operators. These will
prove useful in applications of DLs.

B. Reasoning With DLs

Although the original goal of DLs was to provide a conve-
nient form for expressing the desired knowledge and inferences
of some application, a highly influential paper [22], explored
the idea that choosing a subset of concept constructors leads
to description logics of more restricted expressiveness, but at
the same time more efficient reasoning. As a result, there is a large
body of literature considering various combinations and vari-
a tions of constructors for which reasoning is decidable, or even
tractable. We present in Table II just a small sampling of the
known results about the complexity of computing subsumption
for various combinations of term constructors. The reader may
also find interesting reference [51], which describes interesting
connections between DLs and other formalisms. Hence:

The choice of term constructors is tailored to the expressive
purposes at hand, tempered by the desired computational
properties of the resulting reasoner, especially its decidability.

| TABLE II
SOME SUBSUMPTION COMPLEXITY RESULTS |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCEPT CONSTRUCTORS</td>
</tr>
<tr>
<td>and, or, same-as</td>
</tr>
<tr>
<td>ACC, and, all, some, at-most</td>
</tr>
<tr>
<td>and, or, same-as</td>
</tr>
<tr>
<td>and, at-most, at-least, same-as on attributes, fill, one-or</td>
</tr>
</tbody>
</table>

A final note on the implementation of DL reasoners may be
of interest. There are basically two approaches to computing
the subsumption relationship: One is to manipulate descriptions
into a normal form which eliminates certain redundan-
cies, and which makes explicit implied constraints (e.g.,
all(takers, one-of(Am, Bob)) is augmented by at-most
(2, takers)). As a result, when the time comes to compare
two descriptions, it is possible to do so by performing
relatively few operations, usually comparing pairs of subterms
built with the same constructor. This technique has been used
in the implementation of languages such as KANDOR, CLASSIC,
LOOM, and BACK.

A second approach is to reduce the question "Is it the case
that C ⊆ D?" to the question "Is and(C, not(D)) inconsis-
tent?", and then use theorem-proving techniques to answer the
second question. In particular, the implementation of most
"complete" reasoners—ones that find all the inferences san-
tioned by the standard semantics—is based on such an ap-
proach. For example, KRIS uses a tableaux method with rewrite
rules for deciding consistency, which is based on [60], [39].

Having examined the foundations of DLs, we can now turn
to their application to data and knowledge management.

III. USING DLs FOR DATA MODELING

DLs were developed and studied intensively in the field of
Knowledge Representation, so it is not surprising that they are
particularly adept at representing the semantics of real world
situations—including data semantics. In particular, semantic
data models [49], and more recently object-oriented databases
(e.g., [41]), have claimed to capture the meaning of the data
more directly by concentrating on entities (grouped into
classes) related by relationships (often binary).

Suppose we start with a class, such as

```class STUDENT is-a PERSON with
    studNumber : INTEGER;
    level : {1,2,3,4}
```

In terms of DLs, STUDENT and PERSON are primitive con-
cepts, since individual entities need to be asserted as instances
of them—a person cannot be recognized from external proper-
ties alone. The above class declaration then specifies a con-
straint—a necessary condition that must apply to all instances
of STUDENT. This constraint can be expressed, using rela-
tively simple constructors, by requiring STUDENT to be sub-
sumed by the description D =

```and(PERSON,
    all(studNumber, INTEGER),
    at-least(1, studNumber),
    at-most(2, studNumber),
    all(level, one-of(1, 2, 3, 4)),
    at-least(1, level),
    at-most(1, level))
```

Such a constraint is written in the form PERSON ≼ D, and
its meaning is to limit the relational structures I used to inter-
pret any other description to those that satisfy the condition
PERSON' ≼ D'.

If the declaration of class STUDENT also specified that
studNumber is a key, we could encode this as the additional
description

```at-most(1, compose(studNumber,
    inverse(studNumber)))
```

which says that if we look for individuals that have the same
student number as this one, we will find at most one (this par-
ticular individual).

The argument that semantic data models, such as DAPLEX
and Entity-Relationship, can be expressed using relatively
limited DLs, as above, has been presented in several papers,
including [30], [11], [28], and [26]. For example, the later
paper models the entity relationship diagram in Fig. 3 by pos-
ing classes STUDENT, ENROLLMENT, and COURSE and add-
ing the following constraints

```ENROLLMENT ≼
    and(
        all(st, STUDENT) at-least(1, st) at-most(1, st),
        all(crs, COURSE) at-least(1, crs) at-most(1, crs),
        all(st, DATE) at-least(1, st) at-most(1, st))
```

```STUDENT ≼
    and(
        all(inverse(st), ENROLLMENT)
```
at-least(1, inverse(at))
at-most(6, inverse(at))

COURSE $\leq$
and(
all(inverse(crs), ENROLLMENT)
at-least(1, inverse(crs)) at-most
(300, inverse(crs)))

while in [28], n-ary relationships (and associated term constructor) are introduced in order to model directly relations.

```
\begin{center}
\begin{tikzpicture}
  \node (student) at (1,1) {STUDENT};
  \node (enrollment) at (3,1) {ENROLLMENT};
  \node (course) at (5,1) {COURSE};
  \node (date) at (3,-1) {DATE};
  \path[->] (student) edge [bend left] node {at (1,6)} (enrollment);
  \path[->] (enrollment) edge node {crs (1,300)} (course);
  \path[->] (enrollment) edge [bend right] node {when (1-...)} (date);
\end{tikzpicture}
\end{center}
```

Fig. 3. Entity-relationship diagram.

Similarly, [30], [10], and [26] show how the non-procedural aspects of object-oriented database schemas such as $O_2$ can be captured using DLs.

Representing the database schema in a suitable, decidable DL has been argued to have a number of advantages:

- The greater expressive power of some DLs (e.g., the presence of negation, disjunction, co-reference constraints or inverses) makes it possible to capture important additional aspects of the data semantics [26].
- By checking whether $C \Rightarrow \text{NOTHING}$ as a consequence of any particular set of constraints, it is possible to detect whether or not the global set of specifications in some schema force class $C$ to be incoherent—i.e., one can help verify the schema’s consistency [30].
- By using the relative complement operator, it is possible to reduce the redundancy in the schema presentation, so that every class declaration contains only the minimal additional constraints on top of the explicitly named parents from which it inherits [11].
- In object oriented models, the descriptions used as constraints on primitive class names correspond to types, and in data models such as $O_2$ there is a requirement that subclasses have more refined type [41]. The subsumption ordering on descriptions corresponds to type refinement, and hence provably correct subsumption algorithms can be used for type checking [50].

Most significantly, description logics provide the opportunity to introduce and give names not just to primitive classes but also to defined/virtual classes, which are essentially views. For example, we can add to a schema the notion of UNDER_ENROLLED_CLASS—a course with 5 or fewer takers, by adding the definition

```
UNDER_ENROLLED_CLASS $\equiv$
and(COURSE, at-most(5, takers))
```

The new and considerable advantage gained in this case is that the system itself can be charged with organizing these views into a subclass hierarchy—a non-trivial task when there are many views. In particular, given an existing taxonomy of views and primitive classes, a classifier program can be used to find the least subsumer(s) and most general subsumees of any new view.

Federated databases [61], and more generally so-called cooperative information systems, where information from several sources is made accessible to users, form a particularly active area of application for DLs [4], [28], [62], [71]. A key reason for this is that in order to make several pre-existing databases co-operate it is necessary to first express and relate their contents and semantics. As argued above, DLs provide a richly expressive medium for this task.

For example, [28] uses an expressive DL to relate the entities and relationships in the schemas of several databases using constraints of the form $\equiv$ and $\leq$, in the presence of some (but not necessarily complete) global world knowledge. This information can then be used once again to detect incoherence and redundancy in the resulting system (or maybe just its description).

One approach to federated databases is to integrate the schema of the participating databases. [62] provides an approach to schema integration which uses the CandiDe DL as the common/canonical data model. Among others, a human is charged with the heuristic task of creating an attribute hierarchy showing the relationships between attributes appearing in the classes of the various schemata. For example, given several databases at the university, a designer might come up with the following hierarchy (where indentation is used to indicate the tree structure)

```
TOP-ROLE
  person-identifier
  person-name
dbl1-stud-name
dbl2-full-name
  person-number
dbl2-emp#
dbl3-socsec#
course-identifier
```

The (formal and automated) subsumption and disjointness operations on descriptions use this attribute hierarchy to provide a list of class pairs that appear to be candidates for comparison, because they are equivalent, disjoint or overlapping; the system then offers to the human user a variety of operators (including Generalize, Specialize, Delete) that can be used to restructure and integrate the components of the schema.

Another technique for developing the “right” schema is proposed in [9], where one starts with individuals and existing classes, and clusters them into potentially new classes. The algorithm, related to the “least common subsumer” notions introduced in [29], is based on the structure of the class definitions, presented as descriptions.
IV. ADDITIONAL USES OF DLs IN INFORMATION SYSTEMS

Although DLs are natural candidates for describing the schema of databases, there are a number of additional ways in which descriptions can be used to help in managing information. To see this, we introduce a somewhat more formal view of Knowledge Base Management Systems (KBMS)—systems which maintain and reason with models of some application domain. Let us start from Levesque’s functional view of a KBMS [43]: The basic idea is to treat a knowledge base as an abstract object on which one can perform two kinds of operations: TELLs and ASKs. TELLs are used to build or modify the model of the domain being maintained by the KBMS

TELL: $L_{Tell} \times KB \rightarrow KB$

while ASKs retrieve information

ASK: $L_{Query} \times KB \rightarrow L_{Answer}$

The proper specification of a KBMS and its behavior therefore requires the definition of four things:

- $L_{Tell}$: a language for describing what we know about the world;
- $L_{Query}$: a language for describing questions that we wish to learn about;
- $L_{Answer}$: the language in which answers will be phrased;
- Query answering: how answers to queries are related to what has been told to the KBMS.

For example, a reasoner based on First Order Logic (FOL) could be described by setting $L_{Tell} = L_{Ask} = \{ \text{well-formed formulae of FOL} \}, L_{Answer} = \{ \text{Yes, No, Unknown} \}$, and defining the answer to some question Q as Yes (respectively No) iff the conjunction of the facts told the KB so far entail Q (respectively $\neg Q$) according to mathematical logic.

Without loss of generality, and with considerable gain in convenience, we allow a whole host of TELL and ASK operations, each with possibly different associated languages. Experience with building large software systems of all kinds, including knowledge bases, has taught us that it is an error prone process. Some ways in which errors can be more easily detected is to allow named abbreviations, to insist on identifiers being declared (so that simple typographical errors can be detected) and to allow assertions to be made about the valid and invalid states of the knowledge base. For this purpose, we distinguish two special kind of TELL operations, DECLARE and CONSTRAIN. In the FOL case, DECLARE would be used to introduce the predicate names and arities for example, while CONSTRAIN may state so-called integrity constraints, which would not be used deduce answers, only to detect errors in what the system is being told.

The reason for introducing the above terminology is to help make the following point:

*Description languages can be used in any of the languages associated with a KBMS, including $L_{Declare}$, $L_{Constrain}$, $L_{Tell}$, $L_{Ask}$ and $L_{Answer}$. In each of these situations, the logic associated

with the description language(s) in question is used to define what it means to answer a question.*

In retrospect, we have investigated already in Section III the use of DLs in $L_{Declare}$ and $L_{Constrain}$. We continue with the other languages.

A. A Database-Like KBMS

Suppose that we have specified the schema of a small university knowledge base, including primitive concepts PERSONS, STUDENTS, COURSES, SUBJECTS, SCIENCES, and roles has-subject, teaches, taughtBy, age and takes, all but the last of which are (single-valued) attributes. We are now ready to describe the current state of the world. We will first need to tell the database about new individuals, e.g., introduce a new individual, Crs431, by invoking an operator:

Crs431 := creat-ind( ).

Information about such individuals is recorded in the database in two ways: by specifying what classes they belong to (e.g., “Crs431 is a COURSE”), and by specifying their inter-relationships through roles (e.g., “Crs431 is taught by Einstein and is taken by Anna, ...”). For this purpose, we have operations INSERT-IN and FILL-WITH, which are used as follows:

INSERT-IN(Crs431,COURSE)
FILL-WITH(Crs431,taughtBy,Einstein),
FILL-WITH(Crs431,takers,Anna)
...

Suppose that after several such operations we want to retrieve some information, by asking a question. Queries are characterizations of those objects which satisfy their conditions. We have already seen that the natural interpretation of descriptions was as specifications of sets of individuals: if we want to find “All courses with at least 10 students taking it, taught by someone who is in a science department", then the description

and(COURSE,at-least(10,takers),
all(taughtBy,all(in-dept,SCIENCE-DEPT)))

expresses this. The answer to such a query would be a list of individuals that satisfies the conditions of the query—i.e., the ones recognized by the query description. Papers such as [65], [55], [8], [50], and [25] have investigated the use of DLs as query languages.

DLs are particularly useful for querying knowledge bases in situations when the user is not entirely familiar with the contents or structure of the data, or when they are not entirely sure what question they should be asking. The second situation arises in data exploration/mining, which is essentially the activity of looking for interesting correlations or patterns in large sets of data accumulated for other purposes.

In such situations, we find interesting and novel applications of the fact that descriptions can be classified in a subclass hierarchy.

- One can detect incoherent queries—ones which cannot possibly return any individuals because of the semantics of the database—and alert the user that this question is ill-formed.
More generally, in many situations even if a query is coherent, when it returns an empty set as answer, it is a "miss." In such cases, it is reasonable to consider generalizing the query slightly until a non-empty answer set is obtained. The lattice of subsuming descriptions provides the obvious space to search for such generalizations, and therefore the system can provide a helping hand in this task, as illustrated in [3].

The description lattice supports the paradigm of query specification by iterative refinement, described in [65] and [55].

Data exploration involves asking very many queries, possibly by teams of people, over an extended period of time. The DL-based KBMS can automatically organize this large set of queries through the subsumption relationship, thereby allowing users to find identical or similar queries asked in the past, together with their answers [24]. This is important if the queries may require a considerably long time to process, or if users associate comments/observations with queries. The operation of classifying a given new description with respect to some set of previously encountered descriptions is in fact standard in all DL-reasoners, with various techniques for doing so surveyed in [46], [67], [7]. But we emphasize that such a set of classified descriptions forms just a finite subpartial-order of the infinite lattice of description terms.

Most modern database management systems provide a facility for giving names to some queries, because users frequently refer to them (e.g., they represent some subset of the data or some reorganization of it). These named queries are called views in the database world. Such queries may even be "materialized"—i.e., their answers are maintained up-to-date by the KBMS, rather than being evaluated every time someone looks at them. To introduce such views—named and composite concept descriptions—we can use the KBMS operator DECLARE. Queries as descriptions are obviously useful for view definition, with the same advantages detailed above. Moreover, finding that the current query is subsumed by some materialized view may provide a new opportunity for optimization [25], similar to that envisaged for common subexpression analysis for relational queries [38]: one need only test the query predicate on the individuals in the view. In fact, by using again the relative complement operation in the lattice of description, one might find a cheaper test to run on the members of the view.

DLS are naturally suited for expressing queries (i.e., \(L_{\text{Query}}\)) and for defining views (i.e., \(L_{\text{Decl}}\)). The subsumption relationship can be used to automatically organize queries and views into an "is-a" hierarchy through classification, thereby supporting data exploration and query optimization.

Several research issues arise in the use of descriptions for querying databases.

First, although DLS offer a convenient technique for modeling the semantics of an application domain and the semantics of the data, legacy data is usually present in some existing DBMS (at best, a relational one). We must therefore address the issue of retrieving the answer from such databases. One approach, followed in [53], [48], is to model as part of the KB the relations in the database as well as their relationship to the concepts in the semantic model, and then build a component that takes a DL query, transforms it into a query against the DBMS, and returns the answer. Another approach, suitable in cases where there is frequent KB access, or for DLS that are not sufficiently expressive, is to "load" the database into the DL knowledge base. A straightforward approach to this is likely to be have unacceptably poor performance, and [16] offers a way to compile much of the reasoning of the DL classifier into a sequence of SQL queries, thus taking advantage of the bulk processing offered by DBMS.

Second, one must deal with the fact that DLS have limited expressive power. In fact, [70] shows that for all DLS considered so far, even undecidable ones, concepts can essentially be translated to FOL formulas with at most 3 variable symbols. One approach, suggested in [25], is to factor out a "clean" part of the query (for which subsumption reasoning is performed), and put the rest of the query in an opaque, "dirty" box. An alternative, pursued by the LOOM system, is to implement incomplete subsumption reasoning for a very expressive language (which includes FOL as a sublanguage). In either case, note that the approximate nature of the subsumption relationship does not vitiate most of the advantages introduced earlier in this subsection.

B. Using Descriptions in TELLS

Suppose the description language has a constructor such as \(\text{fills}\), which is used in a description like \(\text{fills(\text{age, }40)}\) to describe the class of individuals that have 40 among their fillers for the role \text{age}. Then the operation \(\text{fill-with(\text{Crs431,akers, }Johnny)}\) can be rephrased as \(\text{insert-in(\text{Crs431,fills(takers,Johnny)})}\). This suggests that we might allow associating some arbitrary un-named description with an individual:

\[
\text{insert-in}(\text{New-crs, }\text{and(COURSE, }\text{at-least}(25,\text{akers})\text{all(takers,all(gpa,range(3.1,4.0))})\text{fills(subject, 'AT'})\text{all(taughtBy, fills(department,ComputerSci))})}
\]

This extension, though at first glance quite small, has far-reaching consequences: it allows the KBMS to maintain incomplete information about individuals. For example, in the above case, we do not yet know the exact identity of the person who will teach the course, but we can already gather information about her (e.g., that her department value is ComputerSci). More significantly, we can say things about all (currently unknown) people who will take the course: they will have gpa in the 3.1 to 4.0 range. This information can be used in query processing: when a query like "Find all courses taught by persons in science departments." is stated, then New-crs can be returned if question answering includes checking whether the descriptor of an individual is subsumed by the query.
To assess the significance of this, observe that no database system can represent the kind of indeterminate information provided above about New-crs. Database management systems can currently only handle "null values" for atomic facts such as strings and integers, and they cannot even reason completely about such null values. In contrast, a system such as CLASSIC can represent facts requiring an unbounded number of distinct nulls (e.g., something having at least 15 fillers for a role), and it can still answers its questions correctly and completely in polynomial time. (Of course, CLASSIC does limit the kinds of questions one can ask.)

This expressive power of DLs is also related to a second problematic aspect of databases: so-called "view updates". Because DBMS translate updates to views into updates to the base/primitive concepts from which the views were defined, the set of views that can be updated is extremely restricted. In contrast, asserting in a DL that some individual belongs to a defined concept—a view—is maintained as just another fact about it, and this fact is reasoned with fully. We therefore have

Using DLs in $L_{rel}$ and using subsumption during query processing allows one to assert indefinite information in the knowledge base. This supports, among others, the proper treatment of such traditionally difficult database issues as null values and view updates.

This aspect of KBMS based on DLs may explain in part their success in problems dealing with configuration management [52], [68]: configurations are incomplete designs, which are slowly being built up, yet we want to find out about problems with them before everything is fully known.

C. Using Descriptions in Answers

Traditionally, questions such as "Who teaches New-crs?" or "What is Johnny's age?" are answered by displaying some individual value(s), looked up in the database. The fact that we can associate arbitrary descriptions with individuals allows us to produce easily descriptive answers, representing the terms we have been told or deduced about these values. For the above questions we might now get answers such as and(FACULTY,fills(department,ComputerScI)) or range(19,27).

In fact, this facility is useful not just when there is incomplete information, but also whenever we don't want to return lists, because they are too long for example. It has been argued (e.g., [63]) that in such situations it is appropriate to provide abstract answers. In the case of DL-based KBMS, this can be achieved by finding in the lattice of descriptions the least common subsumer [29] of the set of individuals' descriptions, which captures their commonalities.

Finally, in the case of very large schemas or when users are not fully familiar with the semantics of the domain they are dealing with, it is useful to provide intensional answers to queries: these display what must hold true of any individual (existing or not) that would satisfy the query [17]. The work of Devanbu [32] on Software Information Systems provides one instance where such a facility is useful: when a new software developer joins a team that has been working on some very large project over a long period of time, she may not be aware of the intended structure of the code, which is expressed by many constraints in the schema. By asking for intensional answers, the novice can learn much about this invisible architecture.

We therefore have

Using DLs in $L_{answer}$ provides the ability to give descriptive, abstract or intensional answers, in addition to enumerations of values.

D. Varying the DLs

In order to make it easier for people to learn to use a DL-based KBMS, some systems (e.g., CLASSIC) use the same syntax (i.e., description constructors) in the various languages associated with a KBMS. It is however not necessary to do so. In fact, because of computational costs, it may be desirable to allow different languages for different operators. This should not be too surprising: one can view Relational Databases as KBMS based on First Order Logic, where the $L_{ax}$ contains all formulas, but $L_{rel}$ is restricted to atomic formulas (corresponding to inserting and deleting tuples), while $L_{answer}$ provides only positive atomic formulas.

The approach of varying languages has been advocated in [43], [42], [25] and has been practiced in systems which use DLs as query languages (e.g., [53]).

E. Descriptions as Constraints

We have seen already that it is useful to associate with a primitive concept some necessary conditions that would have to hold of its individual instances. It turns out that such a facility is more widely useful: we might have defined the notion of UNDER-ENROLLED-CLASS as one with at most 5 takers, but it might be a contingent regulation at our university that such courses be allowed only at the senior or graduate level. Such a constraint might be stated using a CONSTR-ay-type operator CONSTR(\langle constrained-set >, \langle constraint-condition >), where both arguments are descriptions. For example, as a result of

\[
\text{CONSTR(UNDER-ENROLLED-CLASS all(level, one-of(4,5))}
\]

whenever a new course individual is added, if it is inconsistent with the constraint description associated with UNDER-ENROLLED-CLASS, an error message would be generated by the system, and the update would not be allowed.

Note that this use of a constraint is more limited than adding a logical implication of the form "If x is an UNDER-ENROLLED-CLASS then x is also an all(level, one-of(Senior,Grad))", because such an implication could be used for deducing new information about individuals, thereby considerably complicating the processing. (This distinction between "integrity checking" rules and "deductive" rules first appeared in deductive databases.)

F. DLs for Stating Rules

A more "active" KBMS can be obtained through the addition of an operation such as ASSERT-RULE(\langle lhs-descrn >, \langle rhs-descrn >), e.g.,
**Borgida: Description Logics in Data Management**

**Assert-Rule:**

course, fills(topic, AI),

**Boring_thing**

This would have the effect that any time an individual is recognized as a course on AI, it would be added to the concept BORING-THING. Such rules were first mentioned in connection with the CONSUL system [47], and have been heavily used in the Loom system [45], as well as other recent systems such as [69] and CLASSIC, while their semantics has been clarified in [36] through the use of “epistemic operators” dealing with the “knowledge” of the system. They are less expressive than standard production rules because their antecedent is often only a single concept (rather than a relationship between individuals) but because of their treatment of incomplete information, rules based on DLs provide other advantages, including [69]:

- classification applied to the antecedent (or even the consequent) of rules can be used to organize them into a hierarchy; this means that the system can help the programmer find closely related rules—a frequent cause of errors in rule-based programming;
- classification can also help implement the usual conflict-resolution strategy of “apply the most specific rule” by using the automatic classifier, rather than relying on the programmer to specify which rule is more specific.

The rules above are not necessarily treated as logical implication—some systems do not “reason with the contrapositive, nor do they do case analysis (e.g., if B(x) ⇒ D(x) and ¬B(x) ⇒ D(x) then always conclude D(a)). One could obviously add rules with different kinds of reasoning strategies: ordinary logical implication, default rules, etc.

In conclusion,

*Descriptions can be used in a natural way to specify a limited set of conditions and actions for a variety of rule languages, including integrity constraints, triggers, defaults, etc. In all such cases, subsumption can be used to organize large sets of such rules, and recognition helps in the firing process.*

V. On the generality of the DL framework

It is important to point out the generality of the above framework. First, there is no reason to restrict the notion of “individual” to mean “object with intrinsic identity”. Therefore, it is entirely possible to consider mathematical entities (e.g., integers, n-tuples), programming language values (e.g., arrays, procedures), composite values (e.g., lists or trees of others kinds of individuals) as individuals, and have descriptions that denote sets of such individuals. Second, there is complete freedom in the choice of term constructors in the language syntax, and their intended interpretation.

Illustrative of the kind of benefits one gains from this freedom are languages for describing actions/plans, and expressing temporal concepts. For example, [12] introduces special concept constructors for describing classes of temporal intervals. Thus

*after(1980) and duration-greater(2, year) and before(now)*

refers to all time intervals beginning after 1980, of duration at least 2 years, which end before the reference time interval now. Such temporal concepts can then be used with constructors sometime and alltime to describe sets of individuals. For example, if we abbreviate the above temporal description as α, then

**Professor and sometime(α, student)**

represents the set of individuals who are professors now and who were students for a period of at least 2 years between 1980 and now.

We have therefore two more observations:

- There is no “universal” set of term constructors. The term constructors used in a DL may be domain or even application specific.

and

*The denotations of concept descriptions need not be atomic individuals, but could have internal (mathematical) structure.*

This is extremely liberating: in talking about courses, there is no obstacle preventing us from developing a new language, or extending an existing language, to talk about domain specific things: for example, if every course has an instructor and a subject, and there is some subtle inference that needs to be performed with these, then we could have a term constructor course(<instructor>,<topic>). (There is a price of course for inventing new constructors—we need to specify how to reason with them and implement this specification!)

VI. Complexity versus expressiveness

We have already mentioned the strong interest in the DL community concerning the decidability and complexity of reasoning with various DLs. The aforementioned complexity results, and the specter of being caught between the Scylla of tractable but inexpressive DL reasoners, and the Charybdis of rich but computationally intractable languages, has elicited a variety of responses concerning the design of DLs and their implementations.

A. Limited Languages

Some authors have argued that DL-based systems need to respond in polynomial time if they are to be useful as “servers” to other problem solvers [22], [54]. This led to a class of languages, including KANDOR and KRYPTON, which had relatively few constructors, carefully chosen so that subsumption would be polynomial-time decidable. This approach has been critiqued [37] on the grounds that if some application needs to make inferences, and the KBMS is not capable of making them, these inferences will be implemented somewhere else, destroying the conceptual coherence of the knowledge base.

B. Complete Reasoners for Intractable Languages

Some researchers [6], [58] feel that as long as the logic is decidable, it is reasonable to deliver to the users a system that reasons correctly with it. The main obstacle faced by this approach is to make the performance of the system be predictable, so that users are aware of the forms of knowledge which
can cause exponential explosion in the time or space used by the system. We remark that certain worst-case complexity results—such as the result that just by allowing definitions can lead to an exponential blow-up during processing [49]—are not considered to be a problem, because the examples are pathological and do not arise in practice.

C. Incomplete Implementations of Logics

Systems such as LOOM explicitly acknowledge to their users that not all inferences sanctioned by the obvious semantics of constructors are implemented. The difficulty faced by this approach is to describe to the user the incompleteness. As we have seen, operational definitions are relatively difficult for DLs. Other kinds of semantic specification techniques have been proposed for this purpose, including non-standard denotational semantics such as those in [56], [19], or proof-theoretic axiomatizations, such as in [13], [14], [57].

D. Providing an “Escape Hatch” in the Language

It is possible to introduce one or more constructors in the language whose semantics are “opaque” for subsumption reasoning, but can still be used for recognizing individuals. For example, CLASSIC’s test-defined concepts are passed a Lisp or C function in order to recognize individuals, but are treated as primitives for subsumption. Such constructors are of course open to abuse, but they have proven to be extremely useful in practical applications of the CLASSIC system.

E. Extensible KBMS Architectures

The idea is to start with a limited language, but when the user runs into its boundaries, she can have them expanded sufficiently to accomplish the task at hand. Note that this usually requires only a subset of the inferences entailed by the obvious semantics of the new constructors, but that this subset might vary from application to application. This approach requires a modular architecture for DL reasoners which is as easy to extend as, for example, a syntax-directed translation scheme used in a programming language compiler. Such extensible architectures are discussed in [15], [5], and the methodology of providing extensions is illustrated in [14].

Our conclusions in this section are that

The conflicting desires between expressive languages and complexity of reasoning, although very real, need not be paralyzing: there is wide variety of approaches to the problem, with the “predictability” of the inferences and their timing being of concern to users.

VII. SUMMARY

Description languages provide a variety of constructors for building terms that can be used to express knowledge about the world. They have found applications in a variety of areas such as data management, linguistics [27], programming languages [2], configuration management [68], [52], and knowledge-based software engineering [33]. DLs exploit their special-purpose constructors in order to provide solutions to such difficult problems as view updates and reasoning with incomplete information. They are therefore an alternative approach to the standard techniques for limiting the expressive power of First Order Predicate Calculus (e.g., Horn-formulas), which rely on the form of the formulas most easily characterizable using the standard logical connectives (negation, disjunction, quantifiers).

At the same time, the framework of DLs is sufficiently flexible to admit with relative ease the introduction of new description constructors, which can be application specific, as illustrated by such systems as CLASP [31]. This allows DLs to be tailored to better serve particular applications.

This survey has attempted to show the utility of DLs in describing the (conceptual) schema of databases. This paper has argued that, contrary to popular myth in AI, DLs are useful not only for defining “terminology”. Descriptions can be used in all the languages associated with a KBMS: for asserting incomplete information about individuals, for obtaining descriptive or intensional answers, for stating rules and constraints, etc.

ACKNOWLEDGMENTS

I am grateful to the members of the CLASSIC team, and especially to Ron Brachman, for their collaboration.

This research was supported in part by Grant IRI 91-19310 from the U.S. National Science Foundation. A portion of this paper is based on an invited presentation given at IFIP ‘92: A. Borgida and P. Devanbu, “Knowledge base management systems using description logics, and their role in software information systems,” Information Processing 92 (vol. 3), pp. 171–181, Elsevier Science Publishers, 1992, and appears with the permission of IFIP and Elsevier Science B.V., Amsterdam, The Netherlands.

REFERENCES

complex object database schemata supporting multiple inheritance,"  


Intelligence: 3rd Congress of IAAI, Springer Verlag LNCS no. 728, 1993.

[13] A. Borgida, "From type systems to knowledge representation: Natural  
semantics specifications for description logics," Int J. of Intelligent  

[14] A. Borgida, "Towards the systematic development of terminological  
reasoners: Clasp reconstructed," Proc. Conf. on Principles of Knowledge  


Alexander Borgida received his PhD degree in computer science from the University of Toronto. He is currently professor of computer science at Rutgers University. His current research interests concern the development of information systems software at the conceptual level, with particular emphasis on language design and associated technical support. Among others, he has investigated topics such as persistent exceptions, requirements modeling, knowledge representation and effective reasoning, and functional specifications. He has collaborated on the Taxis, Daida, and Classic projects and serves on the editorial board of several journals.