Description Logics in Data Management

Alexander Borgida[*†]

October 4, 1995

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.

Keywords: Description, concept, terminological, language, subsumption, knowledge representation, schema, intension, object centered.

1 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
      ...
```

*The author is with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA. E-mail: borgida@cs.rutgers.edu.

†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 [18], and appears with the permission of IFIP and Elsevier Science B.V., Amsterdam.

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

Figure 1: Compositional concept in CLASSIC

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.

1.1 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. Figure 1 contains an example of a typical compositional description, expressed in the CLASSIC language [19]. 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 Figure 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 Figure 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 `AI100` 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 `AI100` are individuals `Calvin` and `Hobbes`, both of which are instances of `GRADS`. Then `AI100` is inferred to be an instance of the description in Figure 1, since all the necessary *and* sufficient conditions of that concept are satisfied.

As a possible clarification of the issues involved, we provide an analogy for those familiar with logic programming[1]. 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 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

$Child(x) \iff (\exists z) ParentOf(z,x)$

$Son(y) \iff (\exists w) Male(y) \wedge ParentOf(w,y)$

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.

[1]This analogy extends an example found in [48]

1.2 Outline

For readers familiar with database management[2], 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 [22, 23], 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 [68, 48]. 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[3].

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.

2 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 [25] 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 Figure 1 in the two other currently most widely used DLs, BACK [46] and LOOM [47], as well as an infix notation used in many theoretical papers:

COURSE **and at-most**(10, takers) **and all**(takers, GRADS)

(:and COURSE (:at-most 10 takers) (:all takers GRADS))

COURSE $\sqcap \leq 10$ takers $\sqcap \forall$takers:GRADS

As proposed Ait-Kaci [1], it is useful to view DLs as special languages obtained by *term composition*. All DLs have (at least) two sorts of terms: *concepts*

[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.

(intuitively, denoting collections of individuals), and *roles* (intuitively denoting relationships between individuals); because functional relationships occur very frequently, such roles are often distinguished, and will be called *attributes* in this paper. Therefore the syntax of DLs consists of rules for creating composite terms from atomic/primitive symbols — identifiers of various sorts — and *term constructors*. For example, suppose **prim** is considered to be a concept term constructor, whose argument identifies a primitive concept, and suppose **all** and **at-most** are also concept constructors, then the following term

and(**prim**(`COURSE`),
  at-most(`20, takers`),
  all(`taughtBy`, **prim**(`PROFESSOR`)))

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:

all(**compose**(`taughtBy,rank`) ,
one-of(`AP,P`))

Table 1 contains a fairly comprehensive list of domain-independent description constructors, from [68], 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 α, corresponding to the description example in this section is

$$\begin{array}{l}
\texttt{COURSE}(\alpha) \\
\wedge\ (\exists x_1 \ldots \exists x_{20})\ \texttt{takers}(\alpha, x_1) \wedge \ldots \wedge \texttt{takers}(\alpha, x_{20}) \\
\wedge(x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots \wedge x_{19} \neq x_{20}) \\
\wedge\ \forall r\ \texttt{taughtBy}(\alpha, r) \supset \texttt{PROFESSOR}(r) \\
\wedge \forall r\ \forall y\ \texttt{taughtBy}(\alpha, r) \wedge \texttt{rank}(r, y) \supset (y = \texttt{AP} \vee y = \texttt{P})
\end{array}$$

It is evident that the encoding in predicate logic is less perspicous, 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

TERM	**INTERPRETATION**
TOP-CONCEPT	$\Delta^{\mathcal{I}}$
NOTHING	$\emptyset$
and[C,D]	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
or[C,D]	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
not[C]	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
all[p,C]	$\{d \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$
some[p,C]	$\{d \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
at-least[n,p]	$\{d \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(d)\rvert \geq n\}$
at-most[n,p]	$\{d \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(d)\rvert \leq n\}$
at-least-c[n,p,C]	$\{d \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(d) \cap C^{\mathcal{I}}\rvert \geq n\}$
at-most-c[n,p,C]	$\{d \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(d) \cap C^{\mathcal{I}}\rvert \leq n\}$
same-as[p,q]	$\{d \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(d) = q^{\mathcal{I}}(d)\}$
subset[p,q]	$\{d \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(d) \subseteq q^{\mathcal{I}}(d)\}$
not-same-as[p,q]	$\{d \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(d) \neq q^{\mathcal{I}}(d)\}$
fills[p,b]	$\{d \in dom^{\mathcal{I}} \mid b^{\mathcal{I}} \in p^{\mathcal{I}}(d)\}$
not-fills[p,b]	$\{d \in dom^{\mathcal{I}} \mid b^{\mathcal{I}} \notin p^{\mathcal{I}}(d)\}$
one-of[b_1,...,b_m]	$\{b_1^{\mathcal{I}}, \ldots, b_m^{\mathcal{I}}\}$

TERM	**INTERPRETATION**
TOP-ROLE	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
IDENTITY	$\{(d,d) \mid d \in \Delta^{\mathcal{I}}\}$
role-and[p,q]	$p^{\mathcal{I}} \cap q^{\mathcal{I}}$
role-or[p,q]	$p^{\mathcal{I}} \cup q^{\mathcal{I}}$
role-not[p]	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$
inverse[p]	$\{(d,d') \mid (d',d) \in R^{\mathcal{I}}\}$
restrict[p,C]	$\{(d,d') \in p^{\mathcal{I}} \mid d' \in C^{\mathcal{I}}\}$
compose[p,q]	$p^{\mathcal{I}} \circ q^{\mathcal{I}}$
product[C,D]	$C^{\mathcal{I}} \times D^{\mathcal{I}}$
trans[p]	$\bigcup_{n>1}(p^{\mathcal{I}})^n$

Table 1:

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 distinguishing 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).*

2.1 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, $D^{\mathcal{I}}$, is empty for every possible relational structure $\mathcal{I}$.
- *Does D subsume C?*: The answer is yes if the denotation of C is a subset of the denotation of D, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, for every possible relational structure $\mathcal{I}$.
- *Are D and C mutually disjoint?*: The answer is yes if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every possible relational structure $\mathcal{I}$.
- *Are D and C equivalent?* The answer is yes if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every possible relational structure $\mathcal{I}$.

The subsumption relationship, which corresponds to material implication between predicates and is symbolized by $\Longrightarrow$, 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 "$D \Longrightarrow$ NOTHING?", while disjointness is answered by "$\mathbf{and}(C, D) \Longrightarrow$ 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 $\Longrightarrow$, to be a mathematical structure called a "meet semi-lattice", 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, 31, 66] 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 $B \Longrightarrow C$, it is possible to consider the notion of *relative complement(s)* in this lattice: the maximal description(s) D such $\mathbf{and}(C, D) \equiv B$.

For example, Figure 2 presents two descriptions, their greatest common subsumee (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**}.

The interested reader may consult [31, 66] 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 subsumption relation $\Longrightarrow$*, and the* **and** *constructor (or its equivalent) 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.*

2.2 Reasoning with DLs

Although the original goal of DLs was to provide a convenient form for expressing the desired knowledge and inferences of some application, a highly influential paper [24], 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 variations of constructors for which reasoning is decidable, or even tractable. We present in Table 2 just a small sampling of the known results about the complexity of computing subsumption for various combinations of term constructors.

```
and(   prim(COURSE),
       at-most(25, takers),
       all(taughtBy, one-of(Gauss,Euclid)))
and(   prim(COURSE), prim(FUNNY-EVENT)
       at-most(20, takers),
       all(taughtBy, one-of(Gauss,Marx)))
```

(a) Two descriptions.

```
and(   prim(COURSE), prim(FUNNY-EVENT),
       at-most(20, takers),
       all(taughtBy, one-of(Gauss)))
```

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

```
and(   prim(COURSE),
       at-most(25, takers),
       all(taughtBy, one-of(Gauss,Euclid,Marx)))
```

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

```
all(taughtBy, one-of(Gauss,Euclid,Marx)))
```

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

Figure 2: Two descriptions and lattice operations on them.

The reader may also find interesting reference [53], 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.

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 redundancies, and which makes explicit implied constraints (e.g., **all**(takers,**one-of**(Ann,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 $\Longrightarrow$ D?" to the question "Is **and**(C,**not**(D)) inconsistent?", 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 sanctioned by the standard semantics — is based on such an approach. For example, KRIS uses a tableaux method with rewrite rules for deciding consistency, which is based on [62, 41].

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

3 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 [42], and more recently object-oriented databases (e.g., [43]), 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 concepts*, since individual entities need to be asserted as instances of them — a person cannot be recognized from external properties alone. The above class declaration then specifies a constraint — a necessary condition that must apply to all instances of STUDENT. This constraint can be expressed, using relatively simple constructors, by requiring STUDENT to be subsumed by the description D $\equiv$

```
and(PERSON,
    all(studNumber , INTEGER),
    at-least(1,studNumber),at-most(1,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 $PERSON \sqsubseteq D$, and its meaning is to limit the relational structures I used to interpret any other description to those that satisfy the condition $PERSON^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

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

CONCEPT CONSTRUCTORS	*ROLE CONSTRUCTORS*	*COMPLEXITY*
and,**all**, **same-as**	—	Undecidable [61]
$\mathcal{ALCNR}$ (**and**, **or**, **not**, **some**, **at-least**, **at-most**)	**role-and**	PSPACE [37, 36]
and, **not**, **all**, **some**, **or**	**compose**, **role-or**, **inverse**, **trans**, **restrict**,TOP-ROLE	EXP-TIME [60]
and, **all**, **at-most**, **at-least**, **same-as** on attributes, **fills**, **one-of** with integers		Polynomial [21]

Table 2: *Some subsumption complexity results*

Figure 3: Entity-Relationship Diagram

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 particular 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 [32, 11, 30] and [28]. For example, the later paper models the entity relationship diagram in Figure 3 by positing classes `STUDENT`, `ENROLLMENT` and `COURSE`, and adding the following constraints

ENROLLMENT $\sqsubseteq$
 and(
　　all(st,STUDENT) **at-least**(1,st) **at-most**(1,st)
　　all(crs,COURSE) **at-least**(1,crs) **at-most**(1,crs)
　　all(when,DATE) **at-least**(1,when) **at-most**(1,when))

STUDENT $\sqsubseteq$
 and(
　　all(**inverse**(st) , ENROLLMENT)
　　at-least(1, **inverse**(st)) **at-most**(6, **inverse**(st)))

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

while in [30], n-ary relationships (and associated term constructor) are introduced in order to model directly relations.

Similarly, [32, 10] and [28] 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, coreference constraints or inverses) makes it possible to capture important additional aspects of the data semantics [28].
- By checking whether $C \Longrightarrow$ 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 [32].
- 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 sub-classes have more refined type [43]. The subsumption ordering on descriptions corresponds to type refinement, and hence provably correct subsumption algorithms can be used for type checking [52].

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 ≐
    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 [63], and more generally so-called "co-operative information systems", where information from several sources is made accessible to users, form a particularly active area of application for DLs [4, 30, 64, 13]. A key reason for this is that in order to make several pre-existing databases cooperate 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, [30] uses an expressive DL to relate the entities and relationships in the schemas of several databases using constraints of the form $\doteq$ and $\sqsubseteq$, 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. [64] 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
         db1-stud-name
         db2-full-name
      person-number
         db2-emp#
         db3-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 [31], is based on the structure of the class definitions, presented as descriptions

4 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[4]. Let us start from Levesque's functional view of a KBMS [45]: 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

$$\text{TELL:} \quad \mathcal{L}_{\text{Tell}} \times \text{KB} \rightarrow \text{KB}$$

while ASKs retrieve information

$$\text{ASK:} \quad \mathcal{L}_{\text{Query}} \times \text{KB} \rightarrow \mathcal{L}_{\text{Answer}}$$

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

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

[4] *Nota Bene:* This view is adopted strictly for didactic purposes. Using DLs in a reasoning system in no way commits the developer to such a view.

For example, a reasoner based on First Order Logic (FOL) could be described by setting $\mathcal{L}_{\text{Tell}} = \mathcal{L}_{\text{Ask}} =$ { well-formed formulae of FOL}, $\mathcal{L}_{\text{Answer}} =$ {`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 $\mathcal{L}_{Declare}$ $\mathcal{L}_{Constrain}$, $\mathcal{L}_{Tell}$, $\mathcal{L}_{Ask}$, and $\mathcal{L}_{Answer}$. 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 3 the use of DLs in $\mathcal{L}_{\text{Declare}}$ and $\mathcal{L}_{\text{Constrain}}$. We continue with the other languages.

4.1 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 `takers`, 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 := CREATE-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 [67, 57, 8, 52] and [27] 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 allert 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 [67] and [57].
- 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 [26]. 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 [48, 69, 7]. But we emphasize that such a set of classified descriptions forms just a finite sub-partial-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 [27], similar to that envisaged for common sub-expression analysis for relational queries [40]: 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., $\mathcal{L}_{Question}$) and for defining views (i.e., $\mathcal{L}_{Declare}$). 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 [55, 50], 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 when 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 [18] 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, [16] 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 [27], 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.

4.2 Using descriptions in TELLs

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

```
INSERT-IN(New-crs,
  and(COURSE,
      at-least(25,takers)
      all(takers,all(gpa,range(3.1,4.0))
      fills(subject, 'AI')
      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 `Computer-Sci`). 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 $\mathcal{L}_{Tell}$ 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 [54, 70]: 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.

4.3 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., [65]) 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* [31] 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 [19]. The work of Devanbu [34] 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 $\mathcal{L}_{Answer}$ provides the ability to give descriptive, abstract or intensional answers, in addition to enumerations of values.

4.4 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 $\mathcal{L}_{\mathrm{Ask}}$ contains all formulas, but $\mathcal{L}_{\mathrm{Tell}}$ is restricted to atomic formulas (corresponding to inserting and deleting tuples), while $\mathcal{L}_{\text{Answer}}$ provides only positive atomic formulas.

The approach of varying languages has been advocated in [45, 44, 27], and has been practiced in systems which use DLs as query languages (e.g.,[57]).

4.5 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 CONSTRAIN-type operator CONSTRAIN(`<constrained-set>,<constraint-condition>`), where both arguments are descriptions. For example, as a result of

```
CONSTRAIN(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 a `UNDER-ENROLLED-CLASS` then x is also a **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.)

4.6 DLs for stating rules.

A more "active" KBMS can be obtained through the addition of an operation such as ASSERT-RULE(`<lhs-descrn>,<rhs-descrn>`), e.g.,

```
ASSERT-RULE(and( 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 [49], and have been heavily used in the LOOM system [47], as well as other recent systems such as [71] and CLASSIC, while their semantics has been clarified in [38] 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 [71]:

- 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 $B(x) \supset D(x)$ and $\neg B(x) \supset D(x)$ 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.

5 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!)

6 Complexity vs. 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:

- *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 [24, 56]. 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 [39] 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.

- *Complete reasoners for intractable languages.*
  Some researchers [6, 60] 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 [51] — are not considered to be a problem, because the examples are pathological and do not arise in practice.

- *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 [58, 21], or proof-theoretic axiomatizations, such as in [14, 15, 59].

- *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.

- *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 [17, 5], and the methodology of providing extensions is illustrated in [15].

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.*

7 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 [29], programming languages [2], configuration management [70, 54], and knowledge-based software engineering [35]. 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[33]. 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.

References

[1] H. Ait-Kaci, *A lattice theoretic approach to computation based on a calculus of partially ordered type structures.*, PhD Thesis, University of Pennsylvania, 1984.

[2] H. Ait-Kaci and A. Podelski, "An overview of Life", *Next Generation Information System Technology: Proc. 1st Int. East/West Data Base Workshop*, Springer-Verlag LNCS 504, pp.42–58, 1990

[3] T.W. Anwar, H. Beck and S. Navathe, "Knowledge mining by imprecise querying: a classification-based approach", *Proc. 8th Conference on Data Engineering'*, Tempe, Arizona, February 1992, 622–630.

[4] Y. Arens, C.Y. Chee, C.N. Hsu, and C. Knoblock, "Retrieving and integrating data from multiple information systems", *Int. J. of Intelligent and Cooperative Information Systems 3*(1), 1994.

[5] F. Baader and P. Hanscke, "A scheme for integrating concrete domains into concept languages", *Proc. IJCAI'91*, Australia, August 1991

[6] F. Baader and B. Hollunder, "KRIS: Knowledge representation and inference system", *ACM SIGART Bulletin 2(3)*, June 1991, 8 – 14.

[7] F. Baader, B. Hollunder, B. Nebel, H-J. Profitlich, "An empirical analysis of optimization techniques for terminological representation systems", *Proc. KR'92*, October 1992, Boston, MA.

[8] H. W. Beck, S. K. Gala, and S. B. Navathe, "Classification as a query processing technique in the CANDIDE semantic data model," *Proc. Fifth IEEE International Data Engineering Conference*, February 1989, 572–581.

[9] H. W. Beck, T. Anwar, and S. B. Navathe, "A conceptual clustering algorithm for database schema design", *IEEE Trans. on Knowledge and Data Engineering*, 6(3), June 1994, pp. 396–411.

[10] S. Bergamaschi and B. Nebel, "Automatic building and validation of complex object database schemata supporting multiple inheritance", *Applied Intelligence*, 4(2), 1994, pp.185-204

[11] S. Bergamaschi and C. Sartori, "On taxonomic reasoning in conceptual design", *ACM Trans.on Database Systems 17(3)*, pp. 385–442, 1992.

[12] C. Bettini, "A family of Temporal Terminological Logics", *Advances in Artificial Intelligence: 3rd Congress of IA*AI*, Springer Verlag LNCS No.728, 1993.

[13] J.M. Blanco, A. Illarramendi, A. Goni, "Building a Federated Relational Database System: An Approach using a Knowledge-Based System", *Int'l J. of Intelligent and Cooperative Information Systems*, vol. 3, no. 4, 1994, pp. 415-455

[14] A. Borgida, "From type systems to knowledge representation: natural semantics specifications for description logics," *Int. J. of Intelligent and Cooperative Information Systems 1*(1), 1992.

[15] A. Borgida, "Towards the systematic development of terminological reasoners: CLASP reconstructed", *Proc. Conf. on Principles of Knowledge Representation (KR'92)*, Boston, MA, October 1992.

[16] A. Borgida, "On the relationship between Description Logic and First Order Logic Queries", *Proc. Conf. Information and Knowledge Management*, Gaithersburg, MD., 1994, pp.219-225.

[17] A. Borgida and R. Brachman, " Customizable classification inference in the ProtoDL description management system", *Proc. Conf. Information and Knowledge Management*, Baltimore, MD, November 1992, pp.482–490.

[18] A. Borgida and R. Brachman, "Loading data into description reasoners", *Proc. ACM SIGMOD Conf. on Data Management*, 1993, Washington, DC, pp. 217 – 226.

[19] A.Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick "CLASSIC: a structural data model for objects," *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, June, 1989, pp. 59–67.

[20] 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.

[21] A. Borgida, and P.F. Patel-Schneider, "A semantics and complete algorithm for subsumption in the CLASSIC description logic", *J. of Artificial Intelligence Research*, 1994, pp.277–308.

[22] R. J. Brachman, "A structural paradigm for representing knowledge," Ph.D. Thesis, Harvard University, Division of Engineering and Applied Physics, 1977. Revised version published as *BBN Report No. 3605*, Bolt Beranek and Newman, Inc., Cambridge, MA, May, 1978.

[23] R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, **9**(2), April–June, 1985, pp. 171–216.

[24] R. J. Brachman and H. J. Levesque, "The tractability of subsumption in frame-based description languages," *Proc. AAAI-84*, Austin, TX, August, 1984, pp. 34–37.

[25] R. J. Brachman, R. E. Fikes, and H. J. Levesque, "Krypton: a functional approach to knowledge representation," *IEEE Computer*, Vol. 16, No. 10, October, 1983, pp. 67–73.

[26] R. Brachman, P.Selfridge, L.Terveen, B.Altman, A. Borgida, F. Halper, T.Kirk, A.Lazar, S.McGuiness, L.Resnick, "Knowledge representation support for data archaelogy", *Int. J. of Intelligent and Cooperative Information Systems 2(2)*, June 1993, pp.159–186.

[27] M.Buchheit, M. Jeusfeld, W. Nutt, and M. Staudt, "Subsumption between queries in object-oriented databases", *Information Systems* 19(1), pp.33-54, 1994.

[28] D. Calvanese, M. Lenzerini, and D. Nardi, "A unified framework for class-based representation formalisms", *Proc. Conf. on Principles of Knowledge Representation (KR'94)*, Bonn, Germany, 1994, pp.109–120.

[29] B. Carpenter, *The logic of typed feature structures: applications to unification grammars, logic programs, and cosntraint resolution*, Cambrige University Press, 1992.

[30] T. Catarci and M. Lenzerini, "Representing and using interschema knowledge in cooperative information systems", *Int. J. of Intelligent and Coorperative Information Systems 2(4)*, pp. 375–398, Decembe 1993.

[31] W. Cohen, A. Borgida, and H. Hirsh, "Computing least common subsumers in description logics", *Proc. of AAAI'92*, San Jose, CA., May 1992.

[32] L. Delcambre and K. Davis, "Automatic validation of object-oriented database structures", *Proc. IEEE Data Engineering Conference*, Los Angeles, CA., pp.2–9, 1989.

[33] P. Devanbu and D. Litman, "Plan-based terminological reasoning," *Proc. Conf. on Principles of Knowledge Representation (KR'91)*, Boston, MA, 1991.

[34] P. Devanbu, R. Brachman, P. Selfridge, and B. Ballard, "LaSSIE: A knowledge-based software information system", *Communications of the ACM,34*(5), May 1991.

[35] P. Devanbu and M. Jones, "The use of description logics in KBSE systems", *Proc. 17th Int. Conf. on Software Engineering*, Sorrento, Italy, 1994.

[36] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt, "Tractable concept languages", *Proc. IJCAI'91*, Australia, August 1991, pp. 458-463.

[37] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt, "The complexity of concept languages", *Proc. KR'91*, Boston, MA., 1991, pp. 151-162.

[38] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt, "Adding epistemic operators to concept languages", *Proc. KR'92*, Bonn, 1992, pp.342-353.

[39] J. Doyle, and R. Patil, "Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services", *Artificial Intelligence 48*(3), April 1991, pp.261–298.

[40] S. Finkelstein, "Common expression analysis in database applications", *Proc. ACM SIGMOD Confernce*, Orlando, FL, 1982, pp.235–245.

[41] B. Hollunder, W. Nutt, and M. Schmidt-Schauss, "Subsumption algorithms for concept description languages", *Proc. 9th ECAI*, Stockholm, Aug. 1990, pp.348-353.

[42] R. Hull and R. King, "Semantic database modeling: survey, applications, and research issues", *ACM Computing Surveys* 19(3), September 1987, pp.201–260.

[43] R. Lecluse and P. Richard, "Modeling complex structures in Object-Oriented Databases", *Proc. ACM PODS Conference*, Philadelphia, PA, 1989, pp. 360–367.

[44] M. Lenzerini and A. Schaerf, "Concept languages as query languages", *Proc. AAAI'91*, pp. 471-476.

[45] H. Levesque, "Foundations of a functional approach to knowledge representation", *Artificial Intelligence 23*(2), 1984, pp. 155–212.

[46] K. von Luck,B. Nebel, C. Peltason, and A. Schmiedel, "The anatomy of the BACK System", *KIT (Kunstliche Intelligenz und Textverstehen) - Report 41*, Technical University of Berlin, Jan. 1987.

[47] R.M. MacGregor, "A deductive pattern matcher", in *Proceedings AAAI-87*, St. Paul, Minnesota (1987) 403–408.

[48] R.M. MacGregor, "The evolving technology of classification-based knowledge representation systems", in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, John Sowa editor,Morgan-Kaufman 1991.

[49] W. Mark, "Rule-based inference in large knowledge bases", *Proc. AAAI'80*, August 1980.

[50] E. Mays, C. Apte, J. Griesmer, J. Kastner. "Organizing knowledge in a complex financial domain", *IEEE Expert*, 1987, pp.61–70.

[51] B. Nebel, "Terminological reasoning is inherently intractable", *Artificial Intelligence 43*, 1990, pp.235-249

[52] B. Nebel and C. Peltason, "Terminological reasoning and information management", in D. Karagianis editor, *Information Systems and Artificial Intelligence: Integration Aspects*, Springer-Verlag, 1991, pp.181–212

[53] B. Nebel and G. Smolka, "Attributive description formalisms and the rest of the world", in O. Herzog, C. Rollinger (eds), *Text understanding in LILOG*, Springer Verlag, Berlin, 1991.

[54] B. Owsnicki-Klewe, "Configuration as a consistency maintenance task," in W. Hoeppner, (ed), *Proc. of GWAI-88*, Springer Verlag, 1988, pp. 77–87.

[55] J. Pastor, D. McKay and T. Finin, "View-concepts: knowledge-based access to databases", *Proc. CIKM-92*, Baltimore, MD, 1992, pp. 84–91.

[56] P. F. Patel-Schneider, "Small can be beautiful in knowledge representation", *Proceedings IEEE Workshop on Principles of Knowledge-Based Systems*, Denver, Colorado (1984) 11–16.

[57] P.F. Patel-Schneider, R.J. Brachman, and H.J. Levesque, "ARGON: knowledge representation meets information retrieval," *Proc. First Conf. on Artificial Intelligence Applications*, Denver, CO, December, 1984, pp. 280–286.

[58] P. F. Patel-Schneider, "A four-valued semantics for terminological logics", *Artificial Intelligence* **38** (1989) 319–351.

[59] V. Royer, J. Quantz, "Deriving inference rules for terminological logics", in *Logics in AI, Proc. of JELIA'92*, D. Pearce, G.Wegner (eds), Springer Verlag, 1992, pp.84–105.

[60] K. Schild, "A correspondence theory for terminological logics — preliminary report", *Proc. IJCAI'91*, Sydney, Australia.

[61] M. Schmidt-Schauss, "Subsumption in KL-ONE is undecidable", in *Proceedings KR'89*, Toronto, Canada, May 1989, 421–431.

[62] M. Schmidt-Schauss, and G. Smolka, "Attributive concept descriptions with complements", *Artificial Intelligence Journal, 48(1)* pp.1–26, 1991

[63] A. Sheth and J. Larson, "Federated Database systems for managing distributed, heterogeneous, and autonomous databases", *ACM Computing Surveys 22(3)*, pp.183-236, 1990.

[64] A. Sheth, S. Gala, and S. Navathe, "On automatic reasoning for schema integration", *Int. J. of Intelligent and Cooperative Information Systems*, 2(1), pp. 23–50, 1993.

[65] C-D Shum and R. Muntz, "Implicit representation of extensional answers", in L. Kerscheberg editor, *Proc. Second Int. Conf. on Expert Database Systems*, Benjamin Cummings, 1989, p.497-522.

[66] G. Teege, "Making the difference: a subtraction operation for description logics",*Proc. Conf. on Principles of Knowledge Representation (KR'94)*, Bonn, Germany, 1994, pp.540–550.

[67] F. Tou, M. Williams, R. Fikes, A. Henderson, T. Malone, "RABBIT: An intelligent database assistant", *Proc. AAAI'82.*

[68] W. A. Woods and J. G. Schmolze, "The KL-ONE family," *Computers and Mathematics with Applications 23*(2-5), Special Issue on Semantic Networks in Artificial Intelligence.

[69] W. A. Woods, "Understanding subsumption and taxonomy: a framework for progress", in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, John Sowa editor,Morgan-Kaufman 1991.

[70] J. Wright, E. Weixelbaum, K. Brown, G. Vesonder, S. Palmer, J. Berman, and H. Moore, "A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T Network Systems", *Proc. Conf. Industrial Applications of AI(IAAI93)* pp.183–193, 1992.

[71] J. Yen, R. Neches, and R. MacGregor, "CLASP: integrating term subsumption systems and production systems", *IEEE Transactions on Knowledge and Data Engineering*, 3(1), pp. 25-32, March, 1991.