Description Logics are not just for the `Flightless-Birds`: A New Look at the Utility and Foundations of Description Logics

Alex Borgida
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08904

June 1992

Abstract

This paper presents some of the underlying principles of description logics (also known as terminological logics or KL-ONE-style languages), grounding them in the lattice of terms organized by the so-called "subsumption" relationship. A survey of the increasingly varied uses of description logics, including industrial applications, is presented by considering their role in a number of different operations that one can apply to a knowledge base, including languages for queries, answers, updates, rules, and constraints. Finally, we discuss some of the complexity results related to the logics of descriptions, and survey a spectrum of responses to the many intractability proofs.

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. Database management systems, knowledge representation and reasoning systems, and many object-oriented programming systems, especially ones used for simulation, are just some of the application areas that can be viewed in this light.

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`[1]. 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 `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, includings ones in industry[2].

1.1 What are 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 [Borgida et al. 1989]. 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:

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

Figure 1: Compositional concept in CLASSIC

- *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 infered to be an instance of the concept defined in Figure 1, since all of necessary *and* sufficient the conditions of that concept are satisfied.

[1] In our examples, we follow the convention that classes are written in capital letters, relationships in lower-case, and individual's names have their first letters in capitals.

[2] Evidence of this can be seen in four recent workshops devoted to this topic [Dagstuhl 1991, Berlin 1991, Marina 1992, FallSymp 1992].

- *Reasoning* about the relationship of one description to others, treating them as "intensional" objects. For example, the description in Figure 1 is *subsumed by* (entails) the following 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. Similarly, the description (**at-least** 4 takers) can be inferred to be *inconsistent with* (**all** takers (**one-of** Ross, George, Dan))) since the number of fillers for anything satisfying the latter description is at most three.

1.2 Outline

For readers familiar with database management, and with an interest in deductive databases[3], the paper provides a tutorial and survey of how reasoning with descriptions can enhance the modeling power of the database (i.e., the kinds of knowedge about the world that can be stored) and facilitate the user's interaction with it.

For readers conversant with Artificial Intelligence, DLs are descendants of the influential KL-ONE system [Brachman 1977, Brachman and Schmolze 1985], 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 [Woods and Schmolze 1992, MacGregor 1991]. Therefore our aim is to provide for this audience a novel, systematic look at the various uses to which DLs are being put — a view considerably broader than that usually assumed in Artificial Intelligence[4]. We also propose to briefly look at the formal foundations of DLs, and the range of responses to the daunting problem of computational complexity posed by reasoning with DLs.

We begin by considering syntactic aspects of description languages and touching briefly on their logics. Thereafter, we present a semi-formal view of Information Systems 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. In the second part of the paper we consider briefly semantic foundations of the logics based on description languages, and after presenting some complexity results about the difficulty of reasoning with descriptins we present a spectrum of approaches to dealing with intractable languages.

Throughout the paper we endeavour to summarize the key points as boxed observations.

2 The syntax of DLs

Although the original KL-ONE system supported a graphical notation for representing definitions of concepts, all DLs since the KRYPTON system [Brachman et al. 1983] 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, LOOM [MacGregor 1987] and BACK [von Luck et al. 1987], as well as an infix notation used in m many theoretical papers:

```
COURSE   and atmost(10, takers) and all(takers, GRADS)

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

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

[3] We assume only elementary familiarity with propositional and first order logic.

[4] For this purpose, some elementary familarity with the functionality of relational databases is assumed.

It is useful to view DLs as special languages obtained by *term composition* — a connection which was first made in [Ait-Kaci 1984]. All DLs have (at least) two sorts of terms: *concepts* (intuitively, denoting collections of individuals), and *roles* (intuitively denoting relationships between individuals). Therefore the syntax of DLs consists of rules for creating composite terms from atomic symbols — identifiers of various sorts — and *term constructors.*

For example, the following is a syntax for a very simple DL:

```
< conceptDesc>  :=
        thing   ;; the universal concept, resembling True
      | nothing   ;; the empty concept, resembling False
      | prim(< atomic-concept-id> )   ;; concepts without necessary and sufficient conditions
      | and(< conceptDesc>+)    ;; concept intersection
      | all(< roleDesc> ,< conceptDesc> )   ;; restricting the range of roles
      | at-least(< pos-int> , < roleDesc> )   ;; lower bound on number of fillers
      | at-most(< non-neg-
int> , < multiroleDesc> )   ;; upper bound on the number of fillers

< roleDesc> :=
        < Role-id>
      | compose(< Role-id> ,< roleDesc> ) ;; relation composition
```

The following would be a syntactically valid description in this term language

```
and(prim(COURSE),
    at-most(20, takers),
    all( compose(taughtBy,rank),prim(TENURED-RANKS)))
```

denoting 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 tenured persons (the range of the $taughtBy \circ rank$ binary relation is a subset of the denotation of `TENURED-RANKS`).

It is the case that descriptions can usually be expressed in standard First Order Predicate Calculus, treating concepts as unary predicates, and roles as binary ones. For the preceding example, the corresponding formula would be

$$\begin{gathered}\texttt{COURSE}(this) \wedge \\ (\exists x_1, \exists x_2, ..., \exists x_{20})\ \texttt{takers}(this, x_1) \wedge \texttt{takers}(this, x_2) \wedge ... \wedge \texttt{takers}(this, x_{20}) \\ \wedge \\ (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge ... \wedge x_{19} \neq x_{20}) \\ \wedge \\ \forall t \forall r\ \texttt{taught} - \texttt{by}(this, t) \wedge \texttt{rank}(t, r) \supset \texttt{TENURED} - \texttt{RANKS}(r)\end{gathered}$$

It is evident that the encoding in predicate logic is much more verbose and complex, mostly because of 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[5]. An interesting distinguishing features 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

[5]It is also much more difficult for predicate logic 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.

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

3 Reasoning with Descriptions

We have mentioned earlier that there are two fundamental aspects to the logic of descriptions: the recognition of individuals that satisfy a description, and detecting various judgements, such as subsumption, that relate pairs of descriptions.

As a possible clarification of the issues involved, we provide an analogy for those familiar with standard logic and logic programming[6]. 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).
Parentof(liz,chuck).       Male(chuck).
ParentOf(liz,ann).         Female(ann).
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.

3.1 Lattices: a foundation for reasoning with description

Since concept and role descriptions can easily be thought of as unary and binary predicates, it is natural to consider their ordering by the implication (subsumption) relationship, which we shall denote by the symbol $\Longrightarrow$. Most existing DLs provide (nullary) term constructors **thing** (the universal concept) and **nothing** (the inconsistent concept), as well as some constructor like **and**, whose semantics is that of set intersection. In addition to having been found useful in all applications, these constructors provide the important benefit of turning the (infinite) space of descriptions into 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). The DLs we have encountered in practice are in fact lattices: it is always possible to find a unique least common subsumer for every pair of descriptions. For example, Figure 2 presents two descriptions and their least common subsumer in the language introduced in Section 2. The interested reader may consult [Cohen et al. 1992] for conditions under which the semi-lattice is guaranteed to be a lattice. We also remark that [Ait-Kaci 1984] presented the first investigations concerning lattice-theoretic aspects of a special DL.

For further details of the latice theoretic underpinnings of DLs, see Appendix 1.

To summarize

[6] This analogy extends an example found in [MacGregor 1991]

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


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


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

Figure 2: Two descriptions and their least common subsumer.

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 which can usually be proven to also be a lattice, and which will be seen to provide the basis for defining various interesting logical relationships and operations on DLs.

4 Using DLs in Information Management

A language, even with an entailment relation and some operations defined over it, is just an abstract mathematical entity. The question facing us is how to use this language.

4.1 On specifying an Information Server

To explain our answer, it is useful to adopt a somewhat formal view of Knowledge Base Management Systems (KBMS) — systems which maintain and reason with models of some application domain[7]. Let us start from Levesque's functional view of a KBMS [Levesque 1984]: 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 mainted 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.

[7] *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 ¬Q) in the formal, logical sense.

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.

Note that this enhanced view of KBMS allows us to cleanly and completely capture something like the relational database model: DECLARE introduces the relation and attribute names, while CONSTRAIN specifies the domains of attributes, as well as keys, referential integrity, etc. The insert/delete/update operators are TELLs with their language often restricted to being simple atomic relationships ('tuples'); ASK is described by giving some standard query language for $\mathcal{L}_{\text{Ask}}$ (e.g., SQL), while the language of answers is relations again. Query answering can be defined non-procedurally by appealing to the idea of 'formula satisfaction'.)

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 $\mathcal{L}_{\text{Tell}}$, $\mathcal{L}_{\text{Ask}}$, $\mathcal{L}_{\text{Answer}}$, $\mathcal{L}_{\text{Constrain}}$, as well as $\mathcal{L}_{\text{Declare}}$. In each of these situations, the logic associated with this description language is used to define what it means to answer a question.

In what follows, we will elaborate on this assertion and its implications, by considering the problem of developing a system that helps manage information about courses, teachers, students, etc.

4.2 A simple database-like KBMS

Let us start with a very simple KBMS, whose functionality is weaker than that of an ordinary relational database management system.

We will want introduce individuals, group them into classes and relate them to each other with (binary) relationships.

In order to do so, we will want to tell the KBMS the names of the relationships, and specify that some of them are functional, e.g., `has-subject, teaches, taught-by, age,...`, while others are multi-valued, e.g., `takers`. We will also need some concepts that correspond to natural kinds, such as `PERSONS, STUDENTS, COURSES, SUBJECTS, SCIENCES`, and we will also want to say that `age` applies to `PERSONS` and `STUDENTS`, but not `SUBJECTS`. In database terminology, we want to specify a *schema* for the knowledge base, where the terms to be used in interacting with it are introduced, and their semantics specified. The field of databases has developed the notion of *semantic/conceptual data model* to facilitate this task [Hull and King 1987], with the original intent of semantic data models being to capture the conceptual view of the data for the prospective users, rather than the organization of the data in the database. Partly because of their origin in knowledge representation research for natural language understanding, description languages are eminently suited for presenting such conceptual models.

For example, we would want to specify that `PERSONS` have ages that are integers, and that `STUDENTS` are always `PERSONS`, who also have `advisors` that are `PROFESSORS`. These aspects of the university domain can be captured by declaring `PERSONS` to be a primitive concept with necessary properties that are specified by a description, which would include a subterm such as **and**(**at-least**(1,age),**at-most**(1,age),**all**(age,INTEGER)). Simi-

larly, STUDENTS would be a primitive concept whose necessary properties include the term **and**(PERSON,**all**(advisor,PROFESSOR)).

DLs are in fact generally more expressive than current semantic data models. Special versions of DLs, such as the Entity-Situation model [Bergamaschi et al. 1988], have been proposed as particularly appropriate for developing conceptual database models. DLs have also been proposed as tools for integrating the schema of disparate databases [Beck et al. 1992, Illamarendi et al. 1991], in situations such as heterogeneous databases.

In addition to allowing more aspects of data semantics to be captured, the logic of DLs is also helpful in schema design because one can check the schema for *consistency*, as follows: suppose a concept specification requires that its instances be both PROFESSORS and STUDENTS, but the conjunction of their necessary condition is incoherent (i.e., subsumed by the concept **nothing**, which is the bottom of the lattice); then no individuals can satisfy the conditions of the newly defined concept, so it is not a useful addition to the schema and is likely a mistake. Such consistency checking can be performed automatically by a DL reasoner and can form the foundation of a database schema design assistant.

> DLs are naturally suited for expressing the conceptual structure of a domain, and thus capturing a schema of the knowledge base. The use of DLs in $\mathcal{L}_{\text{Declare}}$ supports automatic detection of incoherent specifications.

Having specified a schema, 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 interrelationships 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,taught-by,Einstein),
FILL-WITH(Crs431,takers, Anna)
...
```

Suppose that after a number of such operations we want to retrieve some information, by asking a question/query. Queries are characterizations of those objects which satisfy their conditions. We have already seen that the natural interpretation of descriptions was as specifications of kinds 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**(taught-by,**all**(in-dept,SCIENCE-DEPT)))

says this (assuming the obvious semantics for the constructors). 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 [Tou et al. 1982], [Patel-Schneider et al. 1984], [Beck et al. 1989] and [Nebel and Peltason 1991] 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 [Naqvi and Tsur 1989], 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 organized in a subsumption lattice. These include detecting incoherent queries (ones which cannot possibly return any individuals because of the semantics of the database), and supporting the paradigm of query specification by iterative refinement, described in [Tou et al. 1982]

and [Patel-Schneider et al. 1984]. 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 (see [Brachman et al. 1992]). This is important if the queries may require a considerably long time to process.

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 [MacGregor 1991, Woods 1991, Baader et al. 1992]. But we emphasize that such a set of classified descriptions forms just a finite sub-partial-order of the infinite lattice of description terms.

Continuing with data exploration, in many situations a query that returns an empty set as answer is a "miss"; it is therefore reasonable to consider generalizing it slightly until a non-empty answer set is obtained. The infinite lattice of possible 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 [Anwar et al. 1992].

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 DEFINE. Queries as DLs are obviously useful for view definition, with the same advantages detailed above.

> DLs are naturally suited for expressing queries (i.e., $\mathcal{L}_{\text{Question}}$) and for defining views (i.e., $\mathcal{L}_{\text{Declare}}$). The subsumption relationship can be used to automatically *organize* queries and definitions into an "is-a" hierarchy through classification.

4.3 A more powerful KBMS: using DLs in TELLs

If the description language has a constructor such as **fills**, 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 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(taught-by, 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. 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 of fixed point numbers. This information can already 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 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 filles 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 $\mathcal{L}_{\text{Tell}}$ and using subsumption during query processing allows one to assert indefinite information in the knowledge base. This allows, among others, for 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 [Owsnicki-Klewe 1988]: 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.

4.4 DLs 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 give back lists, because they are too long for example. It has been argued (e.g., [Shum and Muntz 1988]) that in such situations it is appropriate to provide *abstract answers*. In the case of DL-based KBMS, this can be achieved simply by finding in the lattice of descriptions the *least common subsumer* [Cohen et al. 1992] of the set of individuals; this description captures their commonalaties.

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, showing must hold true of any individual (existing or not) that would satisfy the query [Borgida et al. 1989]. The work of Devanbu [Devanbu 1991] on Software Information Systems based on DLs 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 browsing through the schema, the novice can learn much about this invisible architecture.

We therefore have

> Using DLs in $\mathcal{L}_{\text{Answer}}$ provides the ability to give descriptive, abstract or intensional answers, in addition to enumerations of values.

4.5 Different DLs for different operators

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 $\mathcal{L}_{\text{Ask}}$ contains all formulas, but $\mathcal{L}_{\text{Tell}}$ 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 [Levesque 1984, Lenzerini and Schaerf 1991], and has been practiced in systems which use DLs as query languages (e.g., [Beck et al. 1989, Patel-Schneider et al. 1984]).

4.6 DLs as components of other languages

We need not be limited to using a single description, when telling the system some information.

4.6.1 Constraints

It is often useful to associate with a concept some *necessary conditions* that would have to hold of its individual instances. For example, we might want to restrict COURSEs so that all fillers of the taught-by role are FACULTY, and all takers are STUDENTS. This could be accomplished by having a CONSTRAIN-type operator CONSTRAIN(<constrained-set>,<constraint-condition>), where both arguments are descriptions

```
CONSTRAIN(COURSE,
        and(all(taught-by,FACULTY),all(takers,STUDENT)))
```

Then, whenever a new course individual is added, if its description is inconsistent with the constraint description associated with COURSE, an error message would be generated by the system, and the update would not be allowed.

Note that this use of a constraint is much more limited than adding a logical implication of the form "If x is a COURSE then x is also a **and**(**all**(taught-by,FACULTY),**all**(takers,STUDENT)))" 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.

4.6.2 Rules

A more "active" KBMS can be obtained through the addition of an operation such as ASSERT-RULE(<lhs-description>,<rhs-description>), 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 [Mark 1980], and have been heavily used in the LOOM system [MacGregor 1987], as well as other recent systems such as CLASP [Yen et al. 1991] and CLASSIC.

They are somewhat 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 [Yen et al. 1991]:

- 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 benefit one gains from this freedom is the work on CLASP [Devanbu and Litman 1991], a language for describing plans in Artificial Intelligence. The authors consider, among others, *sequences of specific actions* — called "scenarios" — as individuals occurring in their world, and then describe classes of such individuals through regular expressions constructed from basic action classes. For example, **sequence(action(**`DIAL`**),loop(action(**`RING`**)))** represents any scenario where there was first a dialing action and then 0 or more ringing actions. A portion of the syntax for this language can be reformulated in our term-notation as follows:

< conceptDesc> ::= < objectDesc> | < planDesc>
< planDesc> ::=
action(< ActDesc>)
| **sequence**(< planDesc> ,< planDesc>)
| **alternative**(< planDesc> ,< planDesc>)
| **loop**(< planDesc>)

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 in the above plan domain, we wanted actions to be represented in further detail, as having pre and post-conditions (like STRIP-operators), we could make these be arguments to the term: **action(**`<precondition state concept >, <postcondition state concept >`**)**, so that `DIAL` might be represented as **action(**`PhoneNotInUse, 7DigitsDialed`**)**. Even more specifically, 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 The semantic specification of DLs.

We have suggested that descriptions are terms, that they are organized by the logic of subsumption, and that this logic is used in many different ways as part of question answering in KBMS.

If B and C are descriptions of sort "concept", then we want B to subsume C (written as C $\Longrightarrow$ B) when, in each possible world, every individual in the denotation of C is in the denotation of B; similarly, for descriptions denoting relationships, `r` $\Longrightarrow_{role}$ `s` if every pair of objects related by r is also related by s (e.g., `daughters` $\Longrightarrow_{role}$ `children`). Given a description language, we therefore always start by looking for the subsumption relation between concepts and between roles.

For any specific language, there are many different possible semantics, which in turn can be presented in several different ways:

- Denotational semantics: Start by defining a function $\mathcal{D}$ that associates with every description B a set of values, $\mathcal{D}$(B), in some domain. Then define C $\Longrightarrow$ B iff $\mathcal{D}$(C) is a subset of $\mathcal{D}$(B). Traditionally, such a denotation function shows how the denotation of a term is obtained from the denotation of its subterms. For example,

$$\mathcal{D}(\mathbf{and}(\alpha,\beta)) \equiv \mathcal{D}(\alpha) \cap \mathcal{D}(\beta)$$

  Such a formal specification for DLs was introduced in [Brachman and Levesque 1984], and has been applied to most DLs that have been developed since then.

- Axiomatic semantics: Specify the $\Longrightarrow$ relation by syntactic rules of inference. Natural deduction systems (especially as used in programming languages, e.g., [Kahn 1987]) seem particularly appropriate for this task, as illustrated by the following subsumption rule for **all** restrictions

$$\frac{\vdash \mathrm{C} \Longrightarrow \mathrm{B} \qquad \vdash \mathrm{s} \Longrightarrow_{role} \mathrm{r}}{\vdash \mathbf{all}(\mathrm{r},\mathrm{C}) \Longrightarrow \mathbf{all}(\mathrm{s},\mathrm{B})}$$

  which states that if the restriction on the role is stronger, and/or the role is more general, then the whole term is more specialized. The semantics of the CLASSIC DL was specified in this way in [Borgida 1992a].

- Operational semantics: Develop a program for computing the function `Subsumes?(B,C)`, and then declare that (C $\Longrightarrow$ B) iff this program returns the value `true`. This is not an uncommon approach, and even systems such as Prolog have been described in this way. For DL-based systems, this approach is relatively unappealing because their implementation is quite complex when there are more than a few description constructors interacting in many, sometime subtle ways — i.e., we lack a *simple* computation model, like that of SLD-resolution and unification for Prolog.

Given its origins in set containment, the *sine qua non* of the subsumption relation is that it be a pre-order, i.e., it should be reflexive and transitive.

The only other requirements to keep in mind when defining subsumption (and hence the logical semantics of constructors) are:

- the need to explain to the users of the language how things should be represented with it and how the system will reason with the terms;
- the desire to produce an implementation of the system. There may be reasons to use DLs as pure *specification* languages, but if we want to build systems that perform reasoning with them, we must have some ideas on how to implement them *in the way they were described to the user.*

6.1 Some Issues in the implementation of DL reasoners

We have seen that subsumption is the fundamental logical relationship between descriptions. Generally, DL reasoners attack this problem in one of two ways: One possibility is

to manipulate descriptions into a normal form which eliminates certain redundancies (e.g., **and**(**at-least**(2,p),**at-least**(4,p)) is replaced by **at-leaşt**(4,p)), and which makes explicit implied constraints, so that when the time comes to compare to descriptions, it is possible to do so only by considering pairs of subterms built with the same constructor (so called"structural subsumption"). This technique has been used in the implementation of languages such as KRYPTON, 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, $\neg$ B) inconsistent?", and then use theorem-proving techniques to answer the second question. In particular, the implementation of the KRIS language is based on such an approach, using a tableaux method for deciding consistency [Hollunder and Nutt 1990].

A second aspect common to DL system implementations is the management of the hierarchy of named descriptions (including primitive atoms) which have been introduced in any particular application. In fact, many people identify the technology of Description Languages with such a *classifier* of descriptions. One important question for such a classifier is how to locate the place in the existing hierarchy of some given new description. [Baader et al. 1992] presents experimental evidence on the efficacy of various approaches to this problem. A second issue is how to maintain and accesses the information about the (transitive) IS-A relationship between this finite set of named concepts. For this purpose, [Agrawal et al. 1989] and [Ait-Kaci et al. 1989] provide algorithms and analyses of two different approaches.

6.2 The logic of individual recognition

Subsumption was a relationship between pairs of descriptions; recognition is a relationship between descriptions and individuals.

There are at least two approaches to indicating how descriptions recognize individuals. One is to make individuals themselves into descriptions and then use the subsumption relationship. So, for example, we could add the constructor **indiv**(<ind-id >), so that **indiv**(Crs431) is just another description. An operation such as FILL-WITH(`Crs431,takers, Johnny`)
would be presented as
INSERT-IN(**indiv**(`Crs431`), **fills**(`takers`,**indiv**(`Johnny`))).
One can then present the question "Is `Crs431`, with the above information, recognized by the description **at-least**(`1,takers`)", as the subsumption test

$$\vdash \mathbf{and}(\mathbf{indiv}(\texttt{Crs431}), \mathbf{fills}(\texttt{takers}, \mathbf{indiv}(\texttt{Johnny}))) \Longrightarrow ^{?} \mathbf{at\text{-}least}(1, \texttt{takers})$$

Historically, an approach along these lines was first tried in the KL-ONE system. However, there appear to be a fundamental ontologic difference between individuals and sets, so that we might find it difficult to make sense of judgements such as **indiv**(`Chuck`) $\Longrightarrow$ **indiv**(`Liz`) or "monsters" such as **and**(**indiv**(`Chuck`),**indiv**(`Crs431`)). Such speculative individuals and relationships may however be of interest to philosophers dealing with intensional phenomena.

More recently, the preferred approach has been to introduce individuals as a second type/sort in the language, and also introduce a second judgment, "is an instance of" ($\longrightarrow$), relating individuals and descriptions. Therefore, when one invokes the operation INSERT-IN(`Crs431,COURSE`), one is actually making the assertion

$$\vdash \texttt{Crs431} \longrightarrow \texttt{COURSE}$$

The judgment $\longrightarrow$ has its own logic, dependent on the contents of the database told so far. In order to make the machinery run smoothly, it must obey the following law:

$$\frac{\vdash \mathrm{j} \longrightarrow \mathrm{C} \qquad \vdash \mathrm{C} \Longrightarrow \mathrm{B}}{\vdash \mathrm{j} \longrightarrow \mathrm{B}}$$

so that the individuals recognized by a concept are also recognized by its subsumers, thus respecting the interpretation of "subsumes" as "contains the denotation of".

Since the chief novelty of DLs lies in their reasoning with descriptions, and because of lack of space, we have devoted relatively little time to the recognition aspects in this paper.

7 Complexity vs. expressiveness

A highly influential paper, [Brachman and Levesque 1984], explored the idea that just like various subsets of predicate logic (e.g., Horn logic) can be selected with different computational complexities, so introducing different concept constructors leads to terminological logics of different expressiveness and complexity. This started a flood of results on the complexity of DL reasoning. Among others, it is now known that subsumption in the original KL-ONE language is undecidable [Patel-Schneider 1989b], and that only relatively few sets of constructors have a tractable subsumption tests [Donini et al. 1991]. One of the more startling results is that reasoning about the equality of role-chains (e.g., the `addresses` of my `children` are the same as my `address`) is undecidable [Schmidt-Schauss 1989], but if roles are restricted to functions, it becomes polynomial time [Ait-Kaci 1984].

In some cases, complexity proofs demonstrated that implementations of DL reasoners were incomplete (e.g., [Nebel 1988, Patel-Schneider 1989b]); this has led to a greater awareness of the need for formal proofs demonstrating that systems based on particular DLs do in fact implement the appropriate logic.

The afore-mentioned 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 [Brachman and Levesque 1984, Patel-Schneider 1984]. 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 [Doyle and Patil 1991] 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 [Baader and Hollunder 1991, Schild 1991] 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 [Nebel 1990] — 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 [Patel-Schneider 1989a, Borgida and Patel-Schneider 1992], or proof-theoretic axiomatizations, such as in [Borgida 1992a, Borgida 1992b].
- *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. A variant of this approach would be to allow the introduction of constructors which essentially

"compute their own subsumption" through a user-provided procedure, without relying on the compiler/interpreter of the DL.

- *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 [Borgida and Brachman 1992, Baader and Hanscke 1991], and the methodology of providing extensions is illustrated in [Borgida 1992b].

Our conclusions in this section are that

> The conflicting desires between expressivenss 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.

8 Summary

Description languages provide a variety of constuctors for building variable-free terms that can be used to express knowledge about the world. DLs exploit the limitation to 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[Devanbu and Litman 1991]. This allows DLs to be tailored to better serve particular applications.

Furthermore, contrary to popular myth, DLs are not useful only for defining "terminology" (e.g., `Red-Birds`): We have seen that 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. It is for this reason that we have consciously chosen in this paper to avoid the use of the term "TBox/Terminology Box", introduced in [Brachman et al. 1983], as the site where reasoning with descriptions is carried out.

As a result of the above advantages, KBMS based on description logics, such as CLASSIC [Borgida et al. 1989], LOOM [MacGregor 1987], and BACK [von Luck et al. 1987], are emerging as useful practical tools in a variety of situations, including interfaces and exploration of traditional databases [Beck et al. 1989, Anwar et al. 1992, Brachman et al. 1992], configuration management [Owsnicki-Klewe 1988], and software informatio¡n systems [Devanbu 1991, Mark et al. 1992].

Acknowledgments: I am deeply grateful to Ron Brachman for giving me the opportunity to join him in the Classic adventure and for his collaboration in many aspects of the work discussed here.

References

[Dagstuhl 1991] Second International Workshop on Terminological Logics, May 6-8 1991, Schloss Dagstuhl, Germany. Organized by B. Nebel, C. Peltason, K. von Luck. Position papers available as Technical Report DFKI-D-91-13, DFKI, D-6600 Saarbrucken 11, Germany.

[Berlin 1991] Terminological Logic Users Workshop, October 1991, Berlin, Germany. Organized by C. Peltason, K. von Luck, and C. Kindermann. Position papers available as KIT-Report 95, Technische Universitat Berlin.

[Marina 1992] The Loom Users Workshop, March 23-24, 1992, USC Information Sciences Institute, Marina del Rey, CA. Organized by R. MacGregor.

[FallSymp 1992] "Issues in Description Logics: Users Meet Developers", AAAI Fall Symposium Series, October 1992, Cambridge, Mass.

[Agrawal et al. 1989] Agrawal, R., Borgida, A., and Jagadish, H.V., "Efficient Management of Transitive Relationships in Large Data and Knowledge Bases", *Proc. ACM SIGMOD '89 Conference*, Portland, Oregon, June 1989.

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

[Ait-Kaci et al. 1989] Ait-Kaci, H., Boyer, R., Lincoln, P., and Nasr, R., "Efficient implementation of Lattice Operations", *ACM Trans. on Programming Languages and Systems 11(1)*, (Jan. 1989), 115-146.

[Ait-Kaci and Nasr 1986] Ait-Kaci, H. and Nasr, R., "LOGIN: A Logic Programming Language with Built-In Inheritance", *Journal of Logic Programming 3*, 1986, 187–215.

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

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

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

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

[Beck et al. 1989] Beck, H. W., Gala, S. K., and Navathe, S. B., "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model," *Proc. Fifth IEEE International Data Engineering Conference*, February 1989, 572–581.

[Beck et al. 1992] Beck, H., Anwar, T., Navathe, S., "Clasification through conceptual clustering", submitted for publication.

[Bergamaschi et al. 1988] Bergamaschi, S., S. Lodi and C. Sartori, "Entity-Situation: a model for knowledge representation", *Proc. EDBT'88*, Springer Verlag, LNCS 201.

[Borgida 1992a] Borgida, A., "From Type Systems to Knowledge Representation: Natural Semantics Specifications for Description Logics," *Int. J. of Intelligent and Cooperative Information Systems 1*(1), 1992.

[Borgida 1992b] Borgida, A. "Towards the systematic development of terminological reasoners: CLASP reconstructed", to appear in *Proc. KR'92*, Boston, MA, October 1992.

[Borgida and Brachman 1992] Borgida, A., and Brachman, R., " Customizable Classification Inference in the ProtoDL Description Management System", submitted for publication.

[Borgida et al. 1989] Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. "CLASSIC: A Structural Data Model for Objects," *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, June, 1989, pp. 59–67.

[Borgida and Patel-Schneider 1992] Borgida, A., and Patel-Schneider, P.F., "A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic", manuscript submitted for publication.

[Brachman 1977] Brachman, R. J., "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.

[Brachman and Schmolze 1985] Brachman, R. J., and Schmolze, J. G., "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, **9**(2), April–June, 1985, pp. 171–216.

[Brachman and Levesque 1984] Brachman, R. J., and Levesque, H. J., "The Tractability of Subsumption in Frame-Based Description Languages," *Proc. AAAI-84*, Austin, TX, August, 1984, pp. 34–37.

[Brachman et al. 1983] Brachman, R. J., Fikes, R. E., and Levesque, H. J., "Krypton: A Functional Approach to Knowledge Representation," *IEEE Computer*, Vol. 16, No. 10, October, 1983, pp. 67–73.

[Brachman et al. 1992] Brachman, R., P.Selfridge, L.Terveen, B.Altman, A. Borgida, F. Halper, T.Kirk, A.Lazar, S.McGuiness, L.Resnick, "Knowledge Representation Support for Data Archaelogy", submitted for publication.

[Cohen et al. 1992] Cohen, W., Borgida, A., and Hirsh H., "Computing Least Common Subsumers in Description Logics", *Proc. of AAAI'92*, San Jose, CA., May 1992.

[Davis 1991] Davis, R., "A tale of two knowledge servers", *AI Magazine 12*(3), Fall 1991.

[Devanbu and Litman 1991] Devanbu, P., and Litman, D., "Plan-based Terminological Reasoning," *Proc. KR'91*, Boston, MA, 1991.

[Devanbu 1991] Devanbu, P., Brachman, R., Selfridge, P., and Ballard, B., "LaSSIE: A Knowledge-Based Software Information System", *Communications of the ACM, 34*(5), May 1991.

[Donini et al. 1991] Donini, F., Lenzerini, M., Nardi, D., and Nutt, W., "Tractable Concept Languages", *Proc. IJCAI'91*, Australia, August 1991, pp. 458-463.

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

[Edelman and Owsnicki 1986] Edelman, J., and Owsnicki, B., "Data models in Knowledge Representation Systems: A case study", *GWAI-86 und 2. Ostereichische Artificial Intelligence-Tagung*, pp.69–74, Springer Verlag (Informatik-Fachberichte), 1986

[Hollunder and Nutt 1990] Hollunder, B., and Nutt, W., "Subsumption algorithms for concept languages", Research Report RR-90-04, DFKI/Kaiserlautern, 1990.

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

[Illamarendi et al. 1991] Illamarendi, A., J. Blanco, A.Goni, "A uniform approach to the design of federated systems using BACK", *Proc. Terminological Users Workshop*, Berlin, October 1991, Tech. Report KIT 95, Technische Universitat Berlin.

[Imielinski 1987] Imielinski, T., "Intelligent Query Answering in Rule Based Systems", *Journal of Logic Programming 4*, pp. 229-257, 1987.

[Kahn 1987] Kahn, G. "Natural Semantics", Rapport de Recherche No. 601, INRIA, Sophia Antipolis, France.

[Lenzerini and Schaerf 1991] Lenzerini, M. and Schaerf, A., "Concept Languages as Query Languages", *Proc. AAAI'91*, pp. 471-476.

[Levesque 1984] Levesque, H. 'Foundations of a Functional Approach to Knowledge Representation', *Artificial Intelligence 23*(2), 1984, pp. 155–212.

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

[MacGregor 1987] MacGregor, R. M., "A Deductive Pattern Matcher", in *Proceedings AAAI-87*, St. Paul, Minnesota (1987) 403–408.

[MacGregor 1991] MacGregor, R., "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.

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

[Mark et al. 1992] Mark, W., Tyler, S., McGuire, J., and Schlossberg, J., "Commitment-Based Software Development", to appear in *IEEE Transactions on Software Engineering*, October 1992.

[Naqvi and Tsur 1989] Naqvi, S., and Tsur, S., *A logical language for data and knowledge bases*, New York : Computer Science Press, 1989.

[Nebel 1988] Nebel, B., "Computational complexity of terminologic reasoning in BACK", *Artificial Intelligence 34(3)*, April 1988.

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

[Nebel and Peltason 1991] Nebel, Bernhard and Christof Peltason, "Terminological Reasoning and Information Management", in D. Karagianis editor, **Information Systems and Artificial Intelligence: Integration Aspects**, Springer-Verlag, 1991, pp.181–212

[Owsnicki-Klewe 1988] Owsnicki-Klewe, B., "Configuration as a Consistency Maintenance Task," in W. Hoeppner, editor, *Proc. of GWAI-88*, Springer Verlag, September, 1988, pp. 77–87.

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

[Patel-Schneider et al. 1984] Patel-Schneider, P. F., Brachman, R. J., and Levesque, H. J., "ARGON: Knowledge Representation Meets Information Retrieval," *Proc. First Conf. on Artificial Intelligence Applications*, Denver, CO, December, 1984, pp. 280–286.

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

[Patel-Schneider 1989b] Patel-Schneider, P. F., "Undecidability of subsumption in NIKL", *Artificial Intelligence* **39** (1989)

[Schewe 1989] Schewe, K.D., " Variant Construction using Constraint Propagation techniques over Semantic Networks.", In *Proceedings of the 5th Austrian AI Conference*, Insbruck, 1989, Springer IFB208, pp. 188–197.

[Schild 1991] Schild, K., "A correspondence theory for terminological logics — preliminary report", *Proc. IJCAI'91*, Sydney, Australia.

[Schmidt-Schauss 1989] Schmidt-Schauss, M., "Subsumption in KL-ONE is undecidable", in *Proceedings KR'91*, Toronto, Canada, May 1989, 421–431.

[Shum and Muntz 1988] Shum, C-D, 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.

[Teege 1991] Teege, G., "Representing declarative knowledge for Intelligent Tutoring Systems", PhD Dissertation, Technical University of Munich, 1991.

[Tou et al. 1982] Tou, F., M. Williams, R. Fikes, A. Henderson, T. Malone, "RABBIT: An intelligent database assistant", *Proc. AAAI'82.*

[Woods and Schmolze 1992] Woods, W. A., and Schmolze, J. G., "The KL-ONE Family," *Computers and Mathematics with Applications 23*(2-5), Special Issue on Semantic Networks in Artificial Intelligence.

[Woods 1991] Woods, W.A., "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.

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

9 Lattice properties of subsumption

Most existing DLs also provide (nullary) concept constructors **thing** (the universal concept) and **nothing** (the inconsistent concept), as well as some constructor like **and**, whose semantics is that of concept intersection: i.e., it obeys the axioms

$$\vdash \mathbf{and}(a,a) \equiv a \qquad \vdash \mathbf{and}(a,b) \equiv \mathbf{and}(b,a) \qquad \vdash \mathbf{and}(a,\mathbf{and}(b,c)) \equiv \mathbf{and}(\mathbf{and}(a,b),c)$$

$$\frac{\vdash \mathrm{c} \Longrightarrow \mathrm{d},\ \vdash \mathrm{c} \Longrightarrow \mathbf{and}(\mathrm{e})}{\vdash \mathrm{c} \Longrightarrow \mathbf{and}(\mathrm{d,e})} \qquad \frac{\vdash \mathrm{c} \Longrightarrow \mathrm{e}}{\vdash \mathbf{and}(\mathrm{c},-) \Longrightarrow \mathrm{e}}$$

In addition to having been found useful in most applications, these constructors provide the important benefit of turning the space of concept descriptions (modulo the equivalence relation $\equiv$) into 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)

Unfortunately, there is no guarantee that this semi-lattice will in fact be a lattice, i.e., that every pair of descriptions has a least common subsumer. Consider for example the following DL, with two constructors

$<$ Desc $>$ ::= **int**(n) with n an integer
| **q-range**($\frac{a}{b}$, $\frac{c}{d}$) with $\frac{a}{b}$ and $\frac{c}{d}$ reduced fractions (i.e., a, b, c, d are integers with $gcd(a,b) = gcd(c,d) = 1,\ b \neq 1,\ d \neq 1$)

Suppose the denotation of **int**(n) is the set $\{n\}$, while the denotation of **q-range**($\frac{a}{b}$, $\frac{c}{d}$) is $\{q \mid q$ is a rational number between $\frac{a}{b}$ and $\frac{c}{d}\}$. In this case, there is no unique least common subsumer (join) for the pair of descriptions **int**(2) and **int**(4), because there is an infinite descending chain of subsumers of the form **q-range**($\frac{a}{b}$, $\frac{c}{d}$), where $\frac{a}{b}$ approximates 2 closer and closer from below, while $\frac{c}{d}$ approximates 4 closer and closer from above.

The standard approaches to guaranteeing that the semi-lattice is a lattice are unfortunately unappealing:

- *Adding* **or** *as a term constructor*: although a concept union operator is present in some DLs , this constructor, together with **and** and primitives already allows one to encode propositional logic without negation, and the implication problem for this subset of logic is known to be co-NP hard.
- *Limiting the length of descending chains*: this standard lattice-theoretic trick is not viable because in almost every existing DL one can express the following infinite descending chain of terms:

```
FEMALE

and(FEMALE,all(children,FEMALE))

and(FEMALE,all(children,and(FEMALE,
                            all(children,FEMALE))))

and(FEMALE,all(children,and(FEMALE,
                            all(children,and(FEMALE,
                                             all(children,FEMALE))))))
...

and(FEMALE,all(children,nothing))
```

However, all the DLs we have encountered in practice (including the above example involving only **and**, **all**, **nothing** and atoms) are in fact lattices: it is always possible to find a unique least common subsumer for every pair of descriptions. The interested reader may consult

[Cohen et al. 1992] for conditions under which the semi-lattice is guaranteed to be a lattice. We also remark that [Ait-Kaci 1984] presented the first investigations concerning lattice-theoretic aspects of a special DL.

9.1 Exploiting the subsumption lattice

Once we have descriptions forming a lattice, we can easily define a whole host of operations on descriptions, which have practical uses:

- *Meet* is just the conjunction of concepts.
- *Join* finds the commonalities of descriptions — it is the least common subsumer (*lcs*). Among others, it is useful in learning generalizations from examples and approximate reasoning [Cohen et al. 1992]. In software engineering applications, Schewe [Schewe 1989] has used DLs to find the least common subsumer between some existing piece of code and some new desired variant of it; this generalization describes the commonalities of two procedure variants and is then maintained in the Software Information System as a suitable vehicle for future software reuse.
- *Disjointness*: Two concepts are disjoint if their meet is the bottom of the lattice, **nothing**. Concept disjointness can be used for detecting inconsistency in the knowledge base, e.g., discovering if some individual is being asserted to be both a `COURSE` and a `STUDENT`. It has also been used to build more effective theorem provers, as in LOGIN [Ait-Kaci and Nasr 1986], where descriptions were used as type constraints on Prolog predicate arguments, and descriptor disjointness prevented useless unification attempts.
- *Coherence*: Is it possible for some individual to satisfy the conditions of the description, or is it inconsistent? This is equivalent to testing subsumption by the bottom of the lattice, **nothing**.
- *Relative complements*: Given descriptions α and β such that $\alpha \Longrightarrow \beta$, define $extra(\alpha, \beta)$ to be the most general description δ such that **and**$(\beta, \delta) \equiv \alpha$; such a δ indicates the minimum additional information one needs to know about a β individual, in order for it to be an α individual. For example, the extra information in the second description in Figure 2, compared to the least common subsumer in the same figure, is just the stronger restriction **one-of**`(Gauss,Marx)` on the `taughtBy` role. (Note that if the DL allowed a negation constructor, we could alos have used a description **not**(**fills**`(taughtBy, Euclid)`) .)

  Suppose now that we want to find the differences between two descriptions B and C, e.g., "What is the difference between a `GRAD-STUDENT` and an `UNDERGRAD-STUDENT`?". One possibility is to return the descriptions $extra(C, join(B, C))$ and $extra(B, join(B, C))$; these show how the two concepts differ from their least common subsumer. Teege [Teege 1991] has explored the uses of relative complement in Computer-Aided Instruction.

Note that all these operations take place over the infinite space of all possible descriptions, rather than the finite subset of them explicitly named or mentioned in a particular application.

10 Description logics are not just for the `Flightless-Birds`

(This section is somewhat more polemic in tone and is intended for readers familiar with the AI literature)

The roots of descriptions logics lie in the seminal work of Ronald Brachman on the KR&R system KL-ONE [Brachman 1977]. In contrast to the then-standard semantic networks, which were basically data-structure manipulation packages, KL-ONE emphasized the complex internal *structure* of concepts, built using a small, pre-defined set of "epistemologic primitives". The fundamental relationship between such concepts was "subsumption" — essentially the inverse of entailment.

Following considerable debate about the confusion between assertions (often default ones!), such as "all elephants are gray", and definitions, such as "those elephants that are gray", Brachman, Fikes and Levesque introduced another influential KR&R system, Krypton [Brachman et al. 1983]. This one used a formal version of the formerly graphical notation for KL-ONE-like languages to *define concepts* in a so-called terminology-box (T-box), in which subsumption reasoning ruled; in turn, such definitions were then used to *make assertions about individuals* in the assertion-box (A-box), using some entirely different language. Among others, defined concepts could "recognize" individuals. From this use of structured descriptions came the name *terminological logics*.

A third influential paper [Brachman and Levesque 1984], co-authored by Brachman and Levesque, explored the computational complexity of computing subsmption for various description constructors. This led to the search for "tractable constructor sets", of which relatively few have been found – pretty well everything interesting we wanted to say could be used to say complex things that would require hard reasoning.

The (unintended) effect of the same people being involved in three significant, but not necessarily linked results has been to blur the distinction between the various ideas, and together with other historical developments has lead to a series of what I consider to be miss-conceptions in the general AI community:

- *"DLs are useful only for managing defined terms such as* `BirdsThatFly` *(and such definitions are rarely needed in practice)."* This view is frequently heard from people building expert systems of various kinds, who say that they never encountered the need for stating definitions. As we saw, DLs can be exploited in many other roles than just defining terms, including asking questions, reporting answers, stating rules, etc.
- *"DL-based systems only perform two or three operations such as checking subsumption or disjointness between definitions, or classifying individuals under defined concept."* This view, evident even in review articles such as [Woods and Schmolze 1992], is a result of confusing the mathematical structure of description logics with the particular applications to which these logics have been put so far. The lattice structure forms a formal basis on which an arbitrary number of operations can be defined, and these may have useful applications.
- *"There is some universal DL, of which researchers have implemented or studied some subset or other."* Again, this view is evident in [Woods and Schmolze 1992], as well as [Donini et al. 1991], where an "exhaustive" analysis is attempted for various sets of constructors; it is probably a legacy of Predicate Calculus, where new logical constants are not normally introduced during domain modeling. We have seen that in some applications it is extremely useful to introduce domain-specific constructors.
- *"Users of DL-based systems are ineluctably caught between the Scylla of tractable DL reasoners, which are insufficient for practical applications, and the Charybdis of expressive reasoners, that are either incomplete or slow, both in an unpredictable/ad-hoc way.* This view emerges in such papers as [Doyle and Patil 1991], [Davis 1991] and [Patel-Schneider 1984] . Our response to this dilemma was presented in Section 7.