On the Relationship between Description Logic and Predicate Logic Queries
LCSR-TR-295-A

Alex Borgida*
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903
borgida@cs.rutgers.edu
Tel: 908-249-5062

December 1992
Revised June 1993

Abstract

Description Languages (DLs) are descendants of the KL-ONE [15] knowledge representation system, and form the basis of several object-centered knowledge base management systems developed in recent years, including ones in industrial use. Originally used for conceptual modeling (to define views), DLs are seeing increased use as query languages for retrieving information. This paper, aimed at a general audience that includes database researchers, considers the relationship between the expressive power of DLs and that of query languages based on Predicate Calculus.

We show that all descriptions built using constructors currently considered in the literature can be expressed as formulae of the First Order Predicate Calculus with at most three variable symbols, though we have to allow numeric quantifiers and infinitary disjunction in order to handle a couple of special constructors. Conversely, we show that all first-order queries (formulae with one free variable) built up from unary and binary predicates using at most three variables can be expressed as descriptions. We establish similar results for the subset of FOPC involving two variable symbols.

We also show that certain subsets of constructors can be used to express only formulae in such well-studied subsets of FOPC as conjunctive queries and existential queries. For these languages, we gain ideas for subsumption algorithms from the work in the database literature on the containment problem of existential queries.

The paper also suggests how one can transfer to DLs results from the theoretical database literature about languages based on predicate calculus with recursion.

*This work was conducted in part while the author was on sabbatical leave at AT&T Bell Laboratories, and it was supported in part by NSF Grant IRI 91-19310.

1 Overview

Description Languages (DLs) are descendants of the KL-ONE [15] knowledge representation system and form the basis of several object-centered knowledge base management systems developed in recent years, including ones in industrial use. Originally, they were used for conceptual modeling. Because of their special syntax — descriptions are variable-free terms with a large variety of logical constants/constructors — DLs support reasoning *about* queries and views, and this can be exploited in many applications where access to data is required. We summarize some of these applications of DLs, though for a fuller survey the reader is directed to [11]. The present work is directly motivated by the recent use of DLs as query languages to access data stored in relational databases [16, 2, 23].

There has been considerable research on the complexity of reasoning with descriptions, especially computing the subsumption (containment) relationship between them. Surprisingly little is known about their expressive power as query languages, in comparison with well-known formalisms based on Predicate Calculus (PC). This is all the more regrettable since researchers in the theory of databases have studied intensively various subsets of PC, and much is known about the complexity of both answering queries as well as deciding query containment (equivalent to subsumption for DLs) — the latter being of interest for the purpose of query optimization.

We show that almost all description constructors can be captured in the subset of FOPC using 3 or fewer variable, or variants of this language that allow numeric quantifiers or infinitary disjunction. As a (partial) converse, we show that all first-order queries expressible with three variable symbols can be expressed as descriptions in DLs. We also study the relationship between restricted DLs and the subset of FOPC obtained when only 2 variables are allowed.

The translation of the various description constructors identifies subsets that can be expressed in well-studied subsets of FOPC such as conjunctive queries and existential queries. For these languages, we gain ideas for subsumption algorithms from the work in the database literature on the containment problem of existential queries.

A brief consideration of two DLs that have appeared in the database literature (CLASSIC [12, 16] and CANDIDE [7, 2]) indicates that they can express queries that cannot be stated in Datalog, but cannot express certain first order queries.

Finally, we indicate how one can transfer to DLs results from the theoretical database literature about languages based on predicate calculus with recursion.

2 Descriptions

Consider a database in some object-centered data model such as a semantic data model or an object-oriented one (without methods). In such a database, we have individual objects related to each other by binary relations that can be single- or multi-valued, and the individuals are grouped into classes. Most object-based data models do not support views, and neither object-based nor relation-based DBMS *reason* with views and their inter-relationships. To put it briefly, one distinguishing characteristic of description languages and logics (DLs), is that they provide for the specification of defined classes (views, if you prefer) and support such reasoning, especially about the containment relationship between classes (a.k.a. concepts).

The second distinguishing feature of DLs is that the language they use for specifying definitions is not based syntactically on First Order Predicate Calculus (FOPC) or set theory, but rather on first-order variable-free terms. For example, consider the description in Figure 1. It is constructed from identifiers denoting binary relations (called **roles**) (e.g., `venue`,`players`), individuals (e.g., `Toronto, 1`) and other, previously named or primitive classes (e.g., `GAME, STADIUM`) using de-

and[GAME,
all[venue,STADIUM],
at-most[10, players],
all[players, **fills**[home-town,Toronto]],
all[duration, **one-of**[1,2,3]]]

Figure 1: Composite description for a concept

scription constructors such as **and**, **all**, **at-most**, **fills** and **one-of**. The description in Figure 1 has as intended denotation *"Games which are held in a stadium* (their `venue` role's value must be an instances of concept `STADIUM`),*involving at most 10 players* (the `players` role has at most 10 values/fillers), *all of whom are from Toronto* (all `players` fillers are restricted to have value `Toronto` for attribute `home-town`); *moreover, the duration of the game is one, two or three hours* (the `duration` value is one of 1, 2 or 3).

In some DLs, it is possible to declare a role to be functional, in which case it is called a *feature*. In others, a role need not be atomic: it is possible to have composite descriptions denoting roles, not just concepts. For example, the term

compose`[children, restrict[children,MALE]]`

can be interpreted to denote the relationship normally referred to as *grandsons* in English, since **restrict** takes a binary relation such as `children` and derives the one where all elements in its range are instances of concept `MALE` (hence `sons`), and **compose** corresponds to binary relation composition. We could then have the class whose instances are *"Persons with at least one grandchild who is a doctor"*, provided by the description

`PERSON` **and some**`[ compose[children,children] , DOCTOR]`

The key inferences concerning a description are (1) **recognizing** whether some individual (together with a set of relationships and descriptions ascribed to it) satisfies its conditions; and (2) deciding whether it is entailed by (**subsumes**) another description. For example, the description in Figure 1 would be subsumed by the description **at-most** `[15,players]` because **at-most** `[10,players]` entails **at-most** `[15,players]`, and a conjunction entails each of its conjuncts. Similarly, the earlier `grandsons` description would be subsumed by **compose**`[children, children]`, denoting the *"grandchildren"* relation.

DLs are descendants of the KL-ONE knowledge-representation language, which was used to describe and define concepts (reasoning about individuals was done by treating them as special concepts), especially in natural language understanding applications. Not surprisingly, DLs are therefore useful for capturing the conceptual schema of databases, i.e., as data definition languages. A number of papers have explored this aspect, including [12, 7, 8, 23]; in fact, it is shown in [9] how the Entity-Relationship and DAPLEX semantic models can be expressed in a DL. The benefits of having a logic of descriptions in this case include the ability to automatically organize defined classes into the IS-A hierarchy, and checking the consistency of the class specifications.

More recently, DLs have been proposed and used as *query languages*, thus integrating the data description and data manipulation languages [12, 7, 2, 16]. This has been particularly fruitful in *data exploration* applications, where one or more users, who may not be fully familiar with the database contents, are looking for interesting facts, correlations, etc. The special abilities of DLs lead to a number of advantages:

- Detecting and reporting *incoherence of queries*.

- *Query generalization*: having asked a query that returns no individuals, the system can systematically try to generalize the description until it results in a concept with a non-empty answer [2].

- *Query organization:* When teams of users explore databases over periods of time, it is useful to be able to organize the queries themselves so that one can find similar queries that have been asked in the past [16]. Moreover, users may record observations about the query and its answer, using "meta" objects.

- *Query formulation by refinement:* A user exploring the database can use the classification hierarchy of concepts to refine her queries [34, 39, 7].

- *Intensional query processing:* The general processing strategy of DL queries is to "classify" the query description with respect to the pre-computed views (or saved previous queries), and then only test their instances rather than processing the entire database [7, 32].

- *Schema browsing:* Since concepts in the schema as well as queries can be compared for subsumption, users can explore and discover previously unknown *generic* facts, such as coding standards or constraints in Software Information Systems [20].

Although a lot is known about DLs as data description languages, there has been no systematic study of them as query languages — the object of the present paper.

2.1 The languages and logics of descriptions

Over the years, a considerable variety of DLs have been proposed, studied and implemented (at last count, over 20 of them), each with some set or other of description constructors. Table 1 presents the language $\mathcal{KL}$, which consists of a quite comprehensive list of the constructors considered in the DL literature so far[1]; it comes from recent survey papers [41] and an effort to establish a standard notation and semantics to support knowledge-base interchange [5]. The language of descriptions is obtained recursively by starting from atomic symbols for *primitive* concepts and roles. In the table and elsewhere, we use the symbols $C, D, ...$ to range over concept descriptions, $p, q, ...$ to range over role descriptions, and $a, b, ...$ to denote individuals. The semantics of terms are given denotationally, using the familiar notion of a *structure* $\mathcal{I}$, which assign unary and binary relations over the domain $\Delta^{\mathcal{I}}$ to primitive concepts and roles.

For the interested reader, we indicate in Table 2 the constructors provided by four of the more widely-known DLs, as well as two DLs specifically oriented for database applications.[2]

The basic constructors were discovered empirically, in efforts to express the meaning of English sentences for example, much in the way that semantic data model features were proposed. The reason for the variants and the numerous combinations explored is the expressiveness vs. computational complexity trade-off: as we add more constructors, computing subsumption becomes more difficult, sometimes dramatically. The goal is to have as expressive a language as possible within the bounds of acceptable computational cost. This also explains the apparent redundancy of the constructors in $\mathcal{KL}$: while it may be possible to express **at-least** with **at-least-c**, reasoning with

[1] Note however that one of the strengths of DLs is the ability to introduce additional, *domain-specific constructors*; for example, researchers have considered specialized languages for reasoning about actions, plans, time, etc.

[2] "Industrial strength" languages such as CLASSIC and LOOM also provide a constructor **test**, which takes as argument either a first-order logic formula or an arbitrary C/LISP procedure. The argument of **test** is usually opaque to the DL reasoner as far as subsumption inferences are concerned, but it can be used for individual recognition.

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]	$\{d \in dom^{\mathcal{I}} \mid b^{\mathcal{I}} \in p^{\mathcal{I}}(d)\}$
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]	$\bigcup_{n>0}(p^{\mathcal{I}})^n$

Table 1: *$\mathcal{KL}$: A compendium of concept and role constructors*

Language.	Concept constr.	Role constr.
CLASSIC [12]	**and**, **all**, **at-least**, **at-most**, **fills***, **one-of***, **same-as** on feature chains	—
LOOM [27]	**and**, **or**, **not**, **all**, **some**, **at-least**, **at-most**, **at-least-c**, **at-most-c**, **same-as**, **subset**, **not-same-as**, **fills**, **one-of**	**role-and**, **compose**, **inverse**, **restrict**, **product**, IDENTITY
BACK [?]	**and**, **all**, **at-least**, **at-most**, **same-as**, **one-of**	**role-and**, **product**
KRIS [4]	**and**, **or**, **not**, **all**, **some**, **at-least**, **at-most**, **same-as** on feature chains	**role-and**, **restrict**
CANDIDE [7]	**and**, **or**, **not**, **all**, **fills**, **one-of**, **at-least**, **at-most**, **at-least-c**	—
FL^*_{inv} [9]	**and**, **all**, **at-least**, **at-most**, **one-of***, atomic **not**	**inverse**

Table 2: *A few proposed languages and their constructors*

Concept constr.	Role constr.	Complexity
CLASSIC		Polynomial [14]
KRIS		PSPACE [21]
and,**all**, **same-as**	—	Undecidable [37]
and, **not**, **all**, **some**, **or**	**compose**, **role-or**, **inverse**, **trans**, **restrict**,TOP-ROLE	EXP-TIME [36]
and, **not**, **all**, **some**, **or**	**compose**, **role-or**, **inverse**, **trans**, **restrict**, features, TOP-ROLE	Undecidable [36]

Table 3: *Some subsumption complexity results*

the former is considerably faster. We present in Table 3 just a few of the known results about the complexity of computing subsumption for various DLs[3].

3 Relating descriptions to predicate calculus

There is an obvious similarity between concepts (respectively roles) in DLs, and unary (resp. binary) predicates in predicate calculus. Such a similarity was already exploited by Schmolze&Israel [38] to give a semantics for the original KL-ONE language using λ-calculus. We wish to pursue this similarity, but in order to compare the two formalisms we need a common framework. Since almost all the work on DLs has been carried out in the framework of binary (as opposed to n-ary) roles/relationships, we will side-step the problem of dealing with n-ary predicates by the simple expedient of starting with databases whose schema introduces only unary and binary predicates. The notion of answer is specified model-theoretically: the database is presented as a structure $\mathcal{I}$, and the answer to a query is defined as a subset of the domain $\Delta^{\mathcal{I}}$ (i.e., we are not allowed to create new objects for answers). Since DLs have been used mostly in object-oriented forms of reasoning, we will also simplify matters by assuming that the underlying domain of values is abstract, so that integers, strings, etc. with their orderings and operations are not available. These restrictions lead us to the following:

Definition 1 *A* DL query *over the schema {G1,G2,....} of unary and binary predicates is a concept description restricted to have G_i's as primitive concepts and role names. Given a database (relational structure) $\mathcal{I}$, the answer to a query Q is the set $Q^{\mathcal{I}}$. A* PC query *over the same schema {G1,G2,...} is a well-formed formula Q(x) of FOPC, possibly with extensions for transitive closure, which may have a single free variable x. Given a database (relational structure) $\mathcal{I}$, the answer to a query Q(x) is the set of values d in $\Delta^{\mathcal{I}}$ for which $Q(d)^{\mathcal{I}}$ is true.*

This definition generalizes in the obvious way to queries with two free variables (respectively roles).

For those familiar with DLs, it should be pointed out that the DL "database" above does not contain incomplete information about individuals, and hence the treatment of answers to queries presumes the "closed world assumption" familiar in relational databases, rather than the "open world assumption" normally found in description logics.

With this in mind, we outline a translation from descriptions to formulae of FOPC with one or two arguments. Since primitive concepts and roles are in a one-to-one correspondence with unary

[3]The complexity of many subsets of KRIS has been thoroughly explored in [21], and they range from PTIME to PSPACE-complete.

and binary predicates, their translation is quite simple: primitive concept `C` is represented as $C(x)$, and primitive role `r` as $r(x,y)$. Thereafter, we proceed recursively, as suggested by the entries in Table 4. Actually, to be fully formal we use λ-expressions for the meaning of the descriptions, e.g., `C` is really $\lambda x.C(x)$. This makes the renaming of arguments needed during recursion work properly. However, we ommited the λs to simplify the table, especially since we will later use a more specialized translation that doesn't rely on them at all. Note that this simple translation must introduce new variables whenever a new quantifier appears, in order to avoid spurious capture of variables. For example, without this precaution, the translation of **compose(p,compose(p,p))** would yield $(\exists z)(p(x,z) \wedge (\exists z)(p(z,z) \wedge p(z,y)))$, which is clearly wrong — we wanted $(\exists z_2)(p(x,z_2) \wedge (\exists z_1)(p(z_2,z_1) \wedge p(z_1,y)))$.

It is known that transitive closure cannot be formulated in standard first-order logic. However, following work such as that of Kolaitis & Vardi [25], we can express the **trans** constructor using *infinitary* disjuntion: **trans**`[p]` is just `p` $\vee$ **compose**`[p,p]` $\vee$ **compose**`[p,compose[p,p]]` $\vee \ldots$

Given that all but one of the constructors can be expressed in FOPC, the reader may wonder why we do not dispense with the special syntax of descriptions and use the familiar notation of the predicate calculus. The first reason is human engineering: DL expressions are considerably more succinct, structured and comprehensible than the corresponding "flat" FOPC formulae. The second reason is that the syntax facilitates the recognition of special patterns of FOPC formulas which are amenable to efficient computation. Finally, the freedom to introduce new *logical constants*, which has been found empirically useful in conceptual modeling, has evidently allowed researchers on DLs to discover subsets of FOPC for which containment is decidable/tractable, subsets that were not identified by logicians who deal with the traditional logical constants $\{\forall, \exists, \wedge, \vee, \neg, =, \neq\}$, or with the primitives of relational algebra.

4 DLs and the number of variables in FOPC

The main results of this paper show that there is a relationship between the constructors used in DLs and subsets of PC with limited number of variables. We begin with a definition.

Definition 2 *Let $\mathcal{L}^k_{\omega,\omega}$ be the set of all FOPC formulas with equality expressible using at most k variables.*[4]

Note that one variable may be reused in nested subformulas, as in $(\forall x, y)(P(x,y) \Rightarrow (\exists x)Q(y,x))$. Properties of such these language families have been studied, among others, by Henkin [22], Barwise [6], Immerman [24, 17], and Kolaitis&Vardi [25].

4.1 DLs without trans, at-least and at-most

Our first result shows that almost everything that can be said with DLs, can be said with just a few variables.

Theorem 1 *All concepts and roles expressible in $\mathcal{KL}-\{$**trans**, **at-least**, **at-most**$\}$ can be translated into formulas of $\mathcal{L}^3_{\omega,\omega}$, and if* **compose** *is also excluded, then $\mathcal{L}^2_{\omega,\omega}$ is enough.*

Proof The proof relies on a more careful encoding of the constructors into predicate calculus, where the same variable is reused as much as possible. We will present the translation function

[4] Remember that in our context the arity of predicates is also bounded by two.

TERM	TRANSLATION TO FOPC
TOP-CONCEPT	$True$
NOTHING	$False$
and[C,D]	$C(x) \wedge D(x)$
or[C,D]	$C(x) \vee D(x)$
not[C]	$\neg C(x)$
all[p,C]	$\forall w\ p(x,w) \Rightarrow C(w)$
some[p,C]	$\exists w\ p(x,w) \wedge C(w)$
at-least[n,p]	$\exists z_1, ..., \exists z_n\ p(x,z_1) \wedge \ldots \wedge p(x,z_n) \wedge z_i \neq z_j$
at-most[n,p]	$\forall z_1, ..., z_n, \forall z_{n+1}\ (p(x,z_1) \wedge \ldots \wedge p(x,z_{n+1}))$ $\Rightarrow (z_1 = z_2 \vee \ldots \vee z_n = z_{n+1})$
at-least-c[n,p,C]	$\exists z_1, ..., \exists z_n\ p(x,z_1) \wedge \ldots \wedge p(x,z_n) \wedge$ $z_i \neq z_j \wedge C(z_i)$
at-most-c[n,p,C]	$\forall z_1, ..., z_n, z_{n+1}\ (p(x,z_1) \wedge \ldots \wedge p(x,z_{n+1})$ $\wedge C(z_j)) \Rightarrow (z_1 = z_2 \vee \ldots \vee z_n = z_{n+1})$
same-as[p,q]	$\forall w\ p(x,w) \Leftrightarrow q(x,w)$
subset[p,q]	$\forall w\ p(x,w) \Rightarrow q(x,w)$
not-same-as[p,q]	$\exists w\ \neg(p(x,w) \Leftrightarrow q(x,w))$
fills[p,b]	$p(x,b)$
one-of[b_1,...,b_m]	$x = b_1 \vee \ldots \vee x = b_m$

TERM	TRANSLATION TO PC
TOP-ROLE	$True$
IDENTITY	$x = y$
role-and[p,q]	$p(x,y) \wedge q(x,y)$
role-or[p,q]	$p(x,y) \vee q(x,y)$
role-not[p]	$\neg p(x,y)$
inverse[p]	$p(y,x)$
restrict[p,C]	$p(x,y) \wedge C(y)$
compose[p,q]	$\exists z\ p(x,z) \wedge q(z,y)$
product[C,D]	$C(x) \wedge D(y)$

Table 4: *Translating description constructors to PC*

$\mathcal{T}\langle\rangle$ in several variants that behave as follows: $\mathcal{T}^x\langle\rangle$ makes x be the free variable of the unary predicate it will produce for its argument concept, while $\mathcal{T}^y\langle\rangle$ makes the free variable be y; i.e., for primitive concept C, $\mathcal{T}^x\langle\mathtt{C}\rangle = C(x)$, while $\mathcal{T}^y\langle\mathtt{C}\rangle = C(y)$. In the case of roles, $\mathcal{T}^{x,y}\langle\mathtt{R}\rangle$ produces a predicate $R(x,y)$, while $\mathcal{T}^{y,x}\langle\mathtt{R}\rangle$ produces predicate $R(y,x)$. The translation functions $\mathcal{T}^x\langle\rangle$, $\mathcal{T}^y\langle\rangle$, and $\mathcal{T}^{x,y}\langle\rangle$ are presented in the following two tables. $\mathcal{T}^{y,x}\langle\rangle$ is obtained from $\mathcal{T}^{x,y}\langle\rangle$ by simultaneously exchanging *all* occurences of x and y (whether free or bound).

TERM C	$\mathcal{T}^x\langle\mathtt{C}\rangle$	$\mathcal{T}^y\langle\mathtt{C}\rangle$
TOP-CONCEPT	$True$	$True$
NOTHING	$False$	$False$
and[C,D]	$\mathcal{T}^x\langle\mathtt{C}\rangle(x) \wedge \mathcal{T}^x\langle\mathtt{D}\rangle(x)$	$\mathcal{T}^y\langle\mathtt{C}\rangle(y) \wedge \mathcal{T}^y\langle\mathtt{D}\rangle(y)$
or[C,D]	$\mathcal{T}^x\langle\mathtt{C}\rangle(x) \vee \mathcal{T}^x\langle\mathtt{D}\rangle(x)$	$\mathcal{T}^y\langle\mathtt{C}\rangle(y) \vee \mathcal{T}^y\langle\mathtt{D}\rangle(y)$
not[C]	$\neg\mathcal{T}^x\langle\mathtt{C}\rangle(x)$	$\neg\mathcal{T}^y\langle\mathtt{C}\rangle(y)$
all[p,C]	$\forall y\ \mathcal{T}^{x,y}\langle\mathtt{p}\rangle \Rightarrow \mathcal{T}^y\langle\mathtt{C}\rangle$	$\forall x\ \mathcal{T}^{y,x}\langle\mathtt{p}\rangle \Rightarrow \mathcal{T}^x\langle\mathtt{C}\rangle$
some[p,C]	$\exists y\ \mathcal{T}^{x,y}\langle\mathtt{p}\rangle \wedge \mathcal{T}^y\langle\mathtt{C}\rangle$	$\exists x\ \mathcal{T}^{y,x}\langle\mathtt{p}\rangle \wedge \mathcal{T}^x\langle\mathtt{C}\rangle$
subset[p,q]	$\forall y\ \mathcal{T}^{x,y}\langle\mathtt{p}\rangle \Rightarrow \mathcal{T}^{x,y}\langle\mathtt{q}\rangle$	$\forall x\ \mathcal{T}^{y,x}\langle\mathtt{p}\rangle \Rightarrow \mathcal{T}^{y,x}\langle\mathtt{q}\rangle$
same-as[p,q]	$\forall y\ \mathcal{T}^{x,y}\langle\mathtt{p}\rangle \Leftrightarrow \mathcal{T}^{x,y}\langle\mathtt{q}\rangle$	$\forall x\ \mathcal{T}^{y,x}\langle\mathtt{p}\rangle \Leftrightarrow \mathcal{T}^{y,x}\langle\mathtt{q}\rangle$
not-same-as[p,q]	$\exists y\ \neg(\mathcal{T}^{x,y}\langle\mathtt{p}\rangle \Leftrightarrow \mathcal{T}^{x,y}\langle\mathtt{q}\rangle)$	$\exists x\ \neg(\mathcal{T}^{y,x}\langle\mathtt{p}\rangle \Leftrightarrow \mathcal{T}^{y,x}\langle\mathtt{q}\rangle)$
fills[p,b]	$\exists y\ (y = b) \wedge \mathcal{T}^{x,y}\langle\mathtt{p}\rangle$	$\exists x\ (x = b) \wedge \mathcal{T}^{y,x}\langle\mathtt{p}\rangle$
one-of[b_1,...,b_m]	$x = b_1 \vee \ldots \vee x = b_m$	$y = b_1 \vee \ldots \vee y = b_m$

TERM R	**TRANSLATION** $\mathcal{T}^{x,y}\langle\mathtt{R}\rangle$
TOP-ROLE	$True$
IDENTITY	$x = y$
role-and[p,q]	$\mathcal{T}^{x,y}\langle\mathtt{p}\rangle \wedge \mathcal{T}^{x,y}\langle\mathtt{q}\rangle$
role-or[p,q]	$\mathcal{T}^{x,y}\langle\mathtt{p}\rangle \vee \mathcal{T}^{x,y}\langle\mathtt{q}\rangle$
role-not[p]	$\neg\mathcal{T}^{x,y}\langle\mathtt{p}\rangle$
inverse[p]	$\mathcal{T}^{y,x}\langle\mathtt{p}\rangle$
restrict[p,C]	$\mathcal{T}^{x,y}\langle\mathtt{p}\rangle \wedge \mathcal{T}^y\langle\mathtt{C}\rangle$
compose[p,q]	$(\exists z)(\ (\exists y)(y = z \wedge \mathcal{T}^{x,y}\langle\mathtt{p}\rangle)\ \wedge\ (\exists x)(x = z \wedge \mathcal{T}^{x,y}\langle\mathtt{q}\rangle)\ )$
product[C,D]	$\mathcal{T}^x\langle\mathtt{C}\rangle \wedge \mathcal{T}^y\langle\mathtt{D}\rangle$

It is very significant that the alternating use of $\mathcal{T}^x\langle\rangle$ and $\mathcal{T}^y\langle\rangle$ in nested concept descriptions (such as **all** or **some**), and the use of the equalities $x = z$ and $y = z$ in the translation of **compose**, make it unnecessary to introduce new variables during the translation process. ■

It turns out that the converses of the above results also hold. To begin with, we have

Theorem 2 *Every query in* $\mathcal{L}^2_{\omega,\omega}$ *can be expressed as a description in a language with concept constructors* {TOP-CONCEPT, NOTHING, **and**, **or**, *atomic* **not**, **some**, **fills**}, *role constructors* {**role-and**, **role-or**, *atomic* **role-not**, **product**, **inverse**} *and special constructors* **same-as-**TOP *and* **single** [5].

Proof Suppose the two variables we can use in $\mathcal{L}^2_{\omega,\omega}$ are x and y. In the following, we will use the notation $\phi()$ to describe a formula that does not have any free variables, $\phi(x)$ to name formulas that have only x as a free variable (but x must in fact occur free in the formula!), and $\phi(x,y)$ as formulas with both x and y occuring unbound.

The proof relies on reconstructing all formulas of $\mathcal{L}^2_{\omega,\omega}$ with up to two free variables as descriptions according to the following pattern:

[5] Their definition is **single**[b]≡**one-of**[b] and **same-as-**TOP[p] ≡ **same-as**[TOP-ROLE,p] which is also equivalent to **at-most**[0,**role-not**[p]] and **all**[**role-not**[p], NOTHING]

Form of Ψ	**Notation**	**Corresp. description**
Ψ has 2 free variables x,y.	$\Psi(x,y)$	Role description R_Ψ, such that ${R_\Psi}^{\mathcal{I}} = \{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \Psi^{\mathcal{I}}(a/x, b/y)\ is\ true\}$
Ψ has 1 free variable, x or y.	$\Psi(x)$, $\Psi(y)$	Concept description D_Ψ, such that ${D_\Psi}^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \Psi^{\mathcal{I}}(a)\ is\ true\}$
Ψ has 0 free variables.	$\Psi()$	Depending on context, concept description D_Ψ or role description R_Ψ, denoting in any $\mathcal{I}$ the universal or empty set.

In general, for a formula $\Psi()$ with no free variables, R_Ψ will be equal to **product**$[D_\Psi, D_\Psi]$.

The translation function, which maps a formula Ψ into a description D_Ψ or R_Ψ, is defined inductively. The base cases occur for atomic formulas involving predicates C and P, and constants b, whose translation is provided in the following tables (omitting symmetric situations where x and y are interchanged).

$\Psi()$	D_Ψ
$C(b)$	**all**[**product**[TOP-CONCEPT,**single**[b]], C]
$P(b,b')$	**all**[**product**[TOP-CONCEPT,**single**[b]], **fills**[P,b']]

Ψ(x)	D_Ψ
$C(x)$	C
$P(x,b)$	**fills**[P,b]
$P(b,x)$	**fills**[**inverse**[P],b]
$P(x,x)$	**some**[**role-and**[P,IDENTITY]]
$x = b$	**single**[b]
$x = x$	TOP-CONCEPT

Ψ(x,y)	R_Ψ
$P(x,y)$	P
$P(y,x)$	**inverse**[P]
$x = y$	IDENTITY

For each logical connective and type of argument we need to provide a recursion step. For negation, if α is any formula then $D_{\neg\alpha}$ = **not**$[D_\alpha]$, while $R_{\neg\alpha}$ = **role-not**$[R_\alpha]$.

For conjunction, we have

$D_{\alpha \wedge \Psi()}$ =	**and**$[D_\Psi, D_\alpha]$
$D_{\Phi(x) \wedge \Psi(x)}$ =	**and**$[D_\Phi, D_\Psi]$
$D_{\Phi(y) \wedge \Psi(y)}$ =	**and**$[D_\Phi, D_\Psi]$
$R_{\Phi(y) \wedge \Psi(x)}$ =	**product**$[D_\Psi, D_\Phi]$
$R_{\Phi(y) \wedge \Psi(x,y)}$ =	**role-and**$[R_\Psi$, **product**[TOP-CONCEPT, D_Φ]]
$R_{\Phi(x) \wedge \Psi(x,y)}$ =	**role-and**$[R_\Psi$, **product**$[D_\Phi$,TOP-CONCEPT]]
$R_{\Phi(x,y) \wedge \Psi(x,y)}$ =	**role-and**$[R_\Phi, R_\Psi]$

For disjunction, the only case in which the differences are other than just replacing **and** by **or** is

$D_{\Phi(y) \vee \Psi(x)}$ =	**or**[**product**$[D_\Psi$,TOP-CONCEPT] , **product**[TOP-CONCEPT,D_Φ]]

Finally, for the quantifiers we need the following cases

Formula Ψ	$D_{(\exists x)\Psi}$	$D_{(\exists y)\Psi}$
$\Psi()$	D_{Ψ}	D_{Ψ}
$\Psi(x)$	**some**[TOP-ROLE, D_{Ψ}]	D_{Ψ}
$\Psi(y)$	D_{Ψ}	**some**[TOP-ROLE, D_{Ψ}]
$\Psi(x,y)$	**some**[**inverse**[D_{Ψ}]]	**some**[D_{Ψ}]

and

Formula Ψ	$D_{(\forall x)\Psi}$	$D_{(\forall y)\Psi}$
$\Psi()$	D_{Ψ}	D_{Ψ}
$\Psi(x)$	**same-as-**TOP[**product**[TOP-CONCEPT,D_{Ψ}]]	D_{Ψ}
$\Psi(y)$	**same-as-**TOP[**product**[TOP-CONCEPT,D_{Ψ}]]	D_{Ψ}
$\Psi(x,y)$	**same-as-**TOP[**inverse**[D_{Ψ}]]	**same-as-**TOP[D_{Ψ}]

This concludes the proof.

■

More generally, we have

Theorem 3 *Every query with up to two free variables in $\mathcal{L}^3_{\omega,\omega}$ can be expressed as a description in the language $\mathcal{KL}-\{$**trans**,**at-least**,**at-most**$\}$*

Proof In this case we deal with formulas having possibly three variables: x, y, and z. Once again we define a recursive procedure for translating formulas into descriptions. Formulas Ψ with two or fewer free variables will be translated, as before, into role descriptions R_{Ψ} or concept descriptions D_{Ψ} that have the same extent as the answer produced by the FOPC query Ψ. The translation of formulas of the form $\Psi(x,z)$ or $\Psi(y,z)$ is carried out by the same procedure as for formulas $\Psi(x,y)$. Similarly, the translation of unary formulas of the form $\Psi(y)$ and $\Psi(z)$ is handled by the same procedure as for formulas $\Psi(x)$. The one novelty will be the translation of subformulas $\Psi(x,y,z)$ with three free variables, since we do not have descriptions denoting ternary relations. But the final query formula can have no more than two free variables, so every subformula $\Psi(x,y,z)$ must in fact be part of a larger subformula of the form $\exists\gamma\Phi(x,y,z)$ or $\forall\gamma\Phi(x,y,z)$, where γ is either x, y or z. Without loss of generality, suppose it is $\exists z\Phi(x,y,z)$. Because all atomic formulas involve only unary or binary predicates, we will be able to prove that it is sufficient to consider the case when $\exists z\Phi(x,y,z)$ is of the form $\exists z\Phi_1(x,z) \wedge \Phi_2(y,z)$, which can be represented by the role description **compose**[R_{Φ_1}, **inverse**[R_{Φ_2}]].

The proof of the intermediate claim, that the case for $\exists z\Phi(x,y,z)$ can be reduced to the form $(\exists z)\Phi_1(x,z) \wedge \Phi_2(y,z)$ relies on putting $\Phi(x,y,z)$ into a normal form that allows the quantifier to be moved in. Specifically, consider the maximal subformulas Ψ_i of $\Phi(x,y,z)$ that have at most two free variables; convert $\Phi(x,y,z)$ into disjunctive normal form $\bigvee_i(\bigwedge_j \Psi_{i,j})$. Then $\exists z\Phi(x,y,z)$ is equivalent to $\bigvee_i \Theta_i$ where $\Theta_i = \exists z(\bigwedge_j \Psi_{i,j})$; but each Θ_i has at most two free variables (z is being quantified over), so Φ can be translated following the previously considered cases, once we have translated each Θ_i. We are therefore left to consider formulas $\exists z\Phi(x,y,z)$, where $\Phi(x,y,z)$ is the conjunction of subformulas Ψ_k, each with at most two free variables. By associativity, group together the subformulas that have the same free variables, thereby obtaining that $\Phi(x,y,z)$ is in the most general case of the form $\Psi_0() \wedge \Psi_1(x) \wedge \Psi_2(y) \wedge \Psi_3(z) \wedge \Psi_4(x,y) \wedge \Psi_5(x,z) \wedge \Psi_6(y,z)$. But then $\exists z\Phi(x,y,z)$ can be rewritten in the form $\beta(x,y) \wedge \exists z(\ (\Psi_3(z) \wedge \Psi_5(x,z)) \wedge \Psi_6(y,z))$. Therefore, in the end the formula in the scope of $\exists z$ does have the desired restricted form, thus concluding the proof. (Note that if formulas Ψ_5 or Ψ_6 are absent from the DNF form of the original formula, then we are left with a formula such as $\exists z\Phi(x,z)$, for which we had provided a recursive translation earlier.)

■

As easy corollaries of the preceding theorems we get

Corollary 1 *The descriptions in* $\mathcal{KL}-\{\mathbf{trans}, \mathbf{compose}, \mathbf{at\text{-}least}(-c), \mathbf{at\text{-}most}(-c)\}$ *express exactly the queries in* $\mathcal{L}^2_{\omega,\omega}$.

The descriptions in $\mathcal{KL}-\{\mathbf{trans}, \mathbf{at\text{-}least}(-c), \mathbf{at\text{-}most}(-c)\}$ *express exactly the queries in* $\mathcal{L}^3_{\omega,\omega}$.

4.2 Numeric quantifiers and transitive closure

The translation of descriptions involving counting, such as **at-least** `[7,children]` into standard FOPC would seem to require 7 distinct variables. We proceed however by a subterfuge, extending the syntax of FOPC to allow numeric/counting quantifiers, as in $(\exists_7 y)\texttt{children}(x,y)$; this predicates the existence of 7 distinct values for which the formula is satisfied. Note that this is not treated as an abbreviation because we wish to say that the above formula has only two variables, x and y!

Let us extend the languages $\mathcal{L}^k_{\omega,\omega}$ with counting quantifiers $\exists_n$, for every positive integer n, and name the resulting languages $\mathcal{L}^k_{\omega,\omega}(\mathbf{COUNT})$. We then have

Theorem 4 *All concepts and roles expressible in* $\mathcal{KL}-\{\mathbf{trans}\}$ *can be translated into formulae of* $\mathcal{L}^k_{\omega,\omega}(\mathbf{COUNT})$.

Proof Simply observe that $\mathcal{T}^x\langle$**at-least-c** `[n,p,C]`$\rangle$ is $(\exists_n y)(\mathcal{T}^{x,y}\langle\texttt{p}\rangle \wedge \mathcal{T}^y\langle\texttt{C}\rangle)$, while **at-most-c** `[n,p,C]` is equivalent to **not** `[`**at-least-c** `[n+1,p,C]]`. ■

Transitive closure is dealt with by introducing infinitary disjunction

$\mathcal{T}^{x,y}\langle$**trans** `[p]`$\rangle = \bigvee_{n=0}^{\infty} \mathcal{T}^{x,y}\langle$**compose**^{n} `[p,p]`$\rangle$

Infinitary disjunction leads to the $\mathcal{L}^k_{\infty,\omega}$ and $\mathcal{L}^k_{\infty,\omega}(\mathbf{COUNT})$. In this case we get the equivalent of Theorems 1 and 4, but not the converses, since there are of course many formulas involving infinite disjunction that cannot be expressed as transitive closure.

4.3 Queries/concepts not expressible in DLs

It is known that there are formulas of FOPC that cannot be expressed with a bounded number of quantifiers. These formulas usually involve predicates over undirected graphs, whose nodes satisfy the undary predicate *Nodes*, and whose edges are represented by the binary predicate *Neighbors*.

To begin with, the results in Immerman [24] show that $\mathcal{L}^{k-1}_{\omega,\omega}$ does not allow the expression of such graph-theoretic properties as the existence of a k-subclique in a graph, or even the fact that the graph consists of k unconnected nodes (i.e., the cardinality of *Nodes* is k and *Neighbors* is the empty relation). In other words, there are pairs of graphs which differ in having the above properties but which cannot be distinguished by any formula of $\mathcal{L}^{k-1}_{\omega,\omega}$.

Therefore, given the concept `NODES` and role `neighbors`, we will have difficulty expressing in $\mathcal{KL}$ even the following simple query from $\mathcal{L}^4_{\omega,\omega}$:

$$(\exists y)(\exists x_1, x_2, x_3)(\bigwedge_i neighbor(y, x_i) \bigwedge_{i \neq j} neighbor(x_i, x_j))$$

This query represents graphs having a 4-connected subcomponent and, by our earlier results, cannot be expressed as a concept in $\mathcal{KL}-\{\mathbf{trans},\mathbf{at\text{-}least},\mathbf{at\text{-}most}\}$. If we disallow **compose** also, then the above query modified to have only variables $\{x_1, x_2, y\}$, rather than $\{x_1, x_2, x_3, y\}$, cannot be represented as a concept either.

The above results also allow us to prove, among others, that **compose** provides an increase in expressive power, which is not entirely obvious since in some situations **compose** can be eliminated; for example **all**[**compose**[p,q],C] is equivalent to **all**[p,**all**[q,C]].

Theorem 5 *The constructor* **compose** *is independent of* { **and**, **or**, **not**, **subset**, **fills**, **one-of**, **role-and**, **role-or**, **role-not**, **restrict**, **product**, **inverse**} *in the sense that it cannot be eliminated by finding an equivalent description without it.*

Proof This is obtained simply from the fact that the query

$$(\exists y)(\exists x_1, x_2)(\bigwedge_i neighbor(y, x_i) \bigwedge_{i \neq j} neighbor(x_i, x_j))$$

can be expressed as a description following the techniques presented in the proof of Theorem 3, which yield in this case the intermediate FOPC formula

$$(\exists x)(\exists y)(neighbors(x, y) \wedge x \neq y \wedge \quad (\exists z)(\ (neighbors(x, z) \wedge x \neq z) \wedge (neighbors(y, z) \wedge y \neq z))\ )$$

whose (rather cryptic) description equivalent is

some[TOP-ROLE, **some**[**role-and**[
role-and[neighbor,**role-not**[IDENTITY]]],
compose[**role-and**[neighbor,**role-not**[IDENTITY]],
inverse[**role-and**[neighbor,**role-not**[IDENTITY]]]]
]]

■

In a more recent paper [17], Immerman et al. present graphs $X(G_k)$ that cannot be distinguished using formulae of $\mathcal{L}^k_{\omega,\omega}$(**COUNT**). These graphs can easily be described using only existential quantifiers and conjunction. The proof of the relevant results relies on certain Ehrenfreucht-Fraisse pebbling games which are shown to correspond to $\mathcal{L}^k_{\omega,\omega}$(**COUNT**). On the other hand, Kolaitis and Vardi [25] present a modified pebbling game that characterizes $\mathcal{L}^k_{\omega,\omega}$(**COUNT**) extended with infinitary disjunction, and the proof in [17] goes through with this new game, thus establishing that transitive closure will not be helpful in expressing these queries. We therefore have that there are queries in $\mathcal{FOL}(\exists, \wedge)$ — the FOPC with only existential quantification and conjunction, that cannot be expressed as descriptions in $\mathcal{KL}$.

5 Existential FOPC queries

Database researchers have studied as query languages for relational databases certain subsets of FOPC, in which universal quantification is disallowed. Datalog (without negation) and the so-called Select-Project-Join relational queries (FOPC with only conjunction and existential quantifiers) are two examples of such restricted families.[6]

The following results provide some insight into the relationship of DLs to the familiar conjunctive and existential FOPC queries that have been studied by the database community.

Lemma 1 *Let* $\mathcal{D}_{conj}$ *be the language obtained with concept constructors* {TOP-CONCEPT, NOTHING, **and**, **some**, **fills**} *and role constructors* {TOP-ROLE, **restrict**, **compose**, **role-and**, **inverse**, **product**}. *Then every description in* $\mathcal{D}_{conj}$ *can be written as a query in* $\mathcal{FOL}(\exists, \wedge)$.

[6]Interestingly, all DLs, which are used for querying object-oriented data models, have included until now the **all** constructor, which provides universal quantification.

The result follows directly by analysis of the translations offered in Table 1. Its most significant consequence is the use for subsumption testing of the algorithm for detecting containment of conjunctive queries ([18] or Chapter 14 of [40]). That algorithm Skolemizes the contained query, and uses it as a database in which to check if the containing query has a non-empty answer. This algorithm can be adapted to DLs

Algorithm 1 *Subsumption Algorithm for* $\mathcal{D}_{conj}$

To determine if F subsumes G, construct a database from G as follows:
All atomic concepts and roles start out empty.
If F and G are concepts then
create a new individual γ and **AssertIn**(γ,G); determine if F recognizes γ;
If F and G are roles then
create new individuals γ_1, γ_2 and **AssertInRole**(γ_1, γ_2,G); determine if F recognizes (γ_1, γ_2);

function **AssertIn**(γ,C)
{ if C is a primitive concept, then add $C(\gamma)$ to the database;
if C=**fills**[p,b], then **AssertInRole**(γ,b,p);
if C=**some**[p,D], then create new individual δ; **AssertInRole**(γ, δ,p); **AssertIn**(δ,D);
if C=**and**[D,E], then **AssertIn**(γ,D); **AssertIn**(γ,E); }

function **AssertInRole**(γ_1, γ_2,R)
{ if R is a primitive role, then add $R(\gamma_1, \gamma_2)$ to the database;
if R=**inverse**[p], then **AssertInRole**(γ_2, γ_1, p);
if R=**role-and**[p,q] then **AssertInRole**(γ_1, γ_2, p);**AssertInRole**(γ_1, γ_2, q);
if R=**compose**[p,q], then create new individual δ, and
AssertInRole(γ_1, δ,p) and **AssertInRole**(δ, γ_2,p) }

Theorem 6 *The above algorithm correctly computes subsumption for* $\mathcal{D}_{conj}$*, and it does so in polynomial time.*

Proof The correctness of the algorithm follows from the fact that it is a translation of the corresponding containment algorithm for formulae in $\mathcal{FOL}(\exists, \wedge)$. To prove the complexity bound, observe that the number k of individuals created in the database by `AssertIn` is clearly bounded by the size of the candidate subsumee, `G`. In order to check whether a description `F` recognize an individual `b` in some database, it is sufficient to compute the extent *extent*(`F`) of the description `F`, and then verify that `b` belongs to *extent*(`F`). The extent of a composite description can be obtained as a relation in a "bottom-up/inside-out" fashion, by first computing the extents of all immediate nested descriptions denoting roles and concepts, and then performing simple relational operations on them, as indicated in the definitions in Table 1. For example, *extent*(**some**`[R,C]`) is computed as π_1(*extent*(`R`) $\bowtie_{2=1}$ *extent*(`C`)) — the join of the second column of table *extent*(`R`) with table *extent*(`C`), followed by projection on the first column. These operations are clearly polynomial in the size of the tables returned for the sub-terms. Since any concept extent can have at most k individuals, and any role extent can have at most k^2 individuals, the computation of *extent*(`F`) will take time polynomial in size(F) + k, which is less than size(F)+ size(G). ■

This result is in contrast with the proven NP-completeness of the containment problem for $\mathcal{FOL}(\exists, \wedge)$, and is connected with the fact that certain conjunctive queries (such as k-clique) cannot be stated in $\mathcal{D}_{conj}$.

Based on results in [35], subsumption in a more expressive language, $\mathcal{FOL}(\exists, \wedge, \vee)$, which also allows disjunction, can be dealt with by putting the query in disjunctive normal form (with a potential exponential blow-up), and then checking the component conjunctive queries for containment as before.

Corollary 2 *Descriptions in $\mathcal{D}_{exist}$, built with constructors in $\mathcal{D}_{conj}$ plus* { **or**, **role-or**} *can be expressed as queries in $\mathcal{FOL}(\exists, \wedge, \vee)$ without equality, and their subsumption is computable in exponential time.*

The subsumption problem for $\mathcal{D}_{exist}$ is certainly CoNP-hard, as it is for any language that can express propositional conjunction and disjunction. The containment problem for existential queries $\mathcal{FOL}(\exists, \wedge, \vee)$ is known to be complete for Π_2, but it is possible that once again the corresponding DL fragment $\mathcal{D}_{exist}$ is less difficult, though still intractable.

Readers familiar with DLs might object that $\mathcal{D}_{conj}$ is uninteresting since one can hardly conceive of a concept definition language without **all** restrictions — **all** and **and** essentially correspond to aggregation and specialization in semantic data models. Our first response is that this might still be useful as a *query* language. Second, because of the nested sub-term structure of descriptions, it may be possible to incorporate such existential terms as "islands" in different languages. Finally, the complement of a concept in $\mathcal{D}_{conj}$ is in fact a language which can be expressed using **all** constructors. The following theorem illustrates a combination of the last two ideas:

Theorem 7 *Consider the language with concept constructors* **and**, **or**, **all**, *primitive* **not**, *primitive* **fills**, *as well as role constructors* **compose**, **inverse**, **role-and**, **role-or**, *primitive* **role-not**, *and* **restrict**, but with the restricting concept being from $\mathcal{D}_{exist}$. *Its subsumption problem can be reduced to that of $\mathcal{D}_{exist}$ augmented with negation of atomic concepts and roles.*

Proof The proof rests on observing that C subsumes D iff **not**[D] subsumes **not**[C], and **not** can be "propagated in" using standard boolean equivalences, plus **not**[**all**[r,E]] = **some**[r,**not**[E]] and **not**[**fills**[p,b]] = **fills**[**not**[p], b]. This results in a concept from $\mathcal{D}_{exist}$, with some atomic concepts and roles possibly negated. ■

6 Expressiveness of implemented DL query systems

A brief remark concerning two DLs, CLASSIC and CANDIDE, that have figured in the database literature as query languages. Since their subsumption (containment) is decidable in sub-exponential time, we have a strong hint that they are not as expressive as FOPC. This is in fact true:

Theorem 8 *The* CLASSIC *and* CANDIDE *DLs cannot express the query $Q(x) = \exists z\ children(x, z) \Leftrightarrow loves(x, z)$, corresponding to the description* **some**`[`**role-and**`[children,loves]]`, *when* `children` *and* `loves` *are not functional. They can however express queries not expressible in Datalog.*

Proof The second result follows from the fact that Datalog is *monotone* — the answer sets of queries cannot decrease when new tuples are added to relations, while the presence of the **all** constructor makes these DLs be non-monotone when new fillers are added to roles. The first result can be also be proven by a form of monotonicity argument: in these two languages, the extent of a description is not altered if some role filler that is an *unnamed* object (i.e., one which is not the denotation of an individual appearing in the description) is replaced by a new individual object which participates in exactly the same relationships as the old one did; this replacement may however invalidate the FOL predication. ■

7 Speculations on Concept Definitions and Recursion

DLs first arose in an attempt to provide facilities for giving *definitions* in knowledge representation systems. They are therefore often used to define "terminology" — new concepts. These new concepts (or roles) can be considered as the equivalents of IDB ("intensional database") predicates in deductive databases, whereas the primitive concepts and roles are EDB ("extensional database") predicates. For example, we might introduce a name for some view, such as `BACHELOR = and[HUMAN, at-most[0,wife]]`, and then use this name in some other concept, such as `STAG-PARTY = and[ all[guests,BACHELOR],...]`.

Such intermediate definitions can be expanded, but they allow one to define concepts quite succinctly so that previously tractable cases of subsumption become CoNP-hard in the presence of terminology definitions [30]. However, such problems do not appear to arise in practice.

Not surprisingly, it is possible to introduce *recursive* definitions, such as

`BINARY-TREE = and[ TREE , at-most[2, branches], all[branches,BINARY-TREE]]`

Note that here the recursion "bottoms out" in cases when an individual has 0 branches, rather than requiring a separate "base case" as is traditional in logic programming. (Such base cases can however be represented in DLs which have the **or** constructor.)

Unfortunately, there are at least three different interpretations for recursive concept definitions [31]: minimal or maximal fixpoints, and "descriptive semantics" (where any fixed point is admitted). Unlike the situation in the Datalog/Prolog community, there is no agreed-on best choice. We summarize again some of the known results in the DL community (see [31] and [3]):

Language	**Semantics**	**Complexity**
and, **all**, **at-most**, **at-least**	gfp, lfp	PSPACE-complete
and, **all**, **same-as** on functional roles, **compose**	fp	Undecidable

The notion of recursion arises in the querying of FOL databases when the same predicate may occur on both sides of a Datalog rule. It is known that the containment problem for Datalog is undecidable. However, a number of special cases have been studied in the literature (e.g., [19, 28]). In particular, when recursion is limited to *monadic* predicates (which is what our concepts are!), it is known that containment is decidable:

Proposition 1 (Cosmadakis et al.) *Given possibly recursive definitions of unary predicates in Datalog*[7]*, query containment is decidable in doubly-exponential time.*

This, and other results (such as those in [1]) might then be applied to description logics, to find other decidable subclasses of recursive definitions. As in the case of conjunctive queries, there exists the possibility of improved time bounds in the case of descriptions.

Acknowledgment

Moshe Vardi started me off on this track by remarking on a possible connection to $\mathcal{L}^2$, and provided some very useful references, as did A. Mendelzon, P. Kolaitis, and others. I had enlightening and valuable discussions with W. Nutt, F.Baader, R. Meyden, M. Yannakakis, P. Devanbu and others. I am very grateful to all of them. The remaining errors in the paper are of course my own.

[7] which uses lfp semantics for recursion

References

[1] Afrati, F., Cosmadakis, S., and M. Yannakakis, "On Datalog vs. Polynomial Time", *Proc. ACM PODS'91*, pp. 13–23.

[2] Anwar, T.M., Beck H. and Navathe S., "Knowledge Mining by imprecise querying: a classification-based approach", *Proc. 8th IEEE Data Engineering Conf.*, Tempe, AZ, February 1992, 622–630.

[3] Baader, F., "Terminological cycles in KLONE-based knowledge representation languages", *Proc. 1991 IJCAI*, SYdney, 1991, pp. 621–626.

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

[5] Baader, F., H-J. Bürckert, J. Heinsohn, B. Hollunder, J. Müller, B. Nebel, W. Nutt, H. Profitlich, *Terminological Knowledge Representation: A Proposal for a Terminological Logic*, DFKI Report, DFKI, Saarbrucken, GERMANY, October 1992.

[6] Barwise, J., "On Moskovakis closure ordinals", *Journal of Symbolic Logic*, 42, 1977, p.292-296.

[7] Beck, H. W., Gala, S. K., and Navathe, S. B., "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model," *Proc. 5th IEEE Data Engineering Conf.*, February 1989, pp. 572–581.

[8] Bergamaschi, S., S. Lodi and C. Sartori, "Entity-Situation: a model for knowledge representation", *EDBT'88.*

[9] Bergamasci, S., Sartori, C., "On taxonomic reasoning in conceptual design", *ACM TODS*, in press. (Tech. Report 78, Dipartimento di Informatica, Universita di Bologna, Italy, January 1991)

[10] Blanco J.M., Illarramendi A., Perez J.M., Goni A., "Making A Federated System Active", *Proc. International Conference on Database and Expert Systems Applications*, Valencia (SPAIN), 1992, A M. Tjoa and I. Ramos eds., Springer-Verlag, pp. 345–350

[11] Borgida, A., "A new look at the foundations and utility of Description Logics (or Terminological Logics are not just for the `Flightless Birds`)", Technical Report, Rutgers University, 1992.

[12] Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. "CLASSIC: A Structural Data Model for Objects," *Proc. 1989 ACM SIGMOD Conf.*, June 1989, pp. 59–67.

[13] Borgida, A., and Brachman, R., "Inference in the Interface to Intelligent and Cooperative Information Systems," *Workshop on Intelligent and Cooperative Information Systems*, October, 1991, Como, Italy.

[14] Borgida, A. and P. Patel-Schneider, *A Semantics and Complete Algorithm for Subsumption in the* CLASSIC *Description Logic*, manuscript, Rutgers University and AT&T Bell Laboratories, February 1992. Submitted for publication.

[15] 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,

[16] Brachman, R., P.Selfridge, L.Terveen, B.Altman, A. Borgida, F. Halper, T.Kirk, A.Lazar, D..McGuiness, L.Resnick, "Knowledge Representation Support for Data Archeology", *Proc. International Conference on Information and Knowledge Management*, Baltimore, MD, November 1992, pp.457–464.

[17] J. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification", *Proc. FOCS 1989*, pp.612–617

[18] Chandra, A., and P. Merlin, "Optimal implementation of conjunctive queries in relational databases", *Proc. ACM STOC'77*, 1977, pp. 77-90.

[19] Cosmadakis, S., H. Gaifman, P. Kanelakis, M. Vardi, "Decidable optimization problems for database logic programs", *Proc. STOC'88*, pp.477–490.

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

[21] Donini, F., Lenzerini, M., Nardi, D., and Nutt, W., "The complexity of concept languages", *Proc. KR'91*, Boston, 1991.

[22] Henkin, L., *Logical systems containing only a finite number of symbols.*, Presses de l'Universite de Montreal, 1967.

[23] Illamarendi, A., Blanco, J. and A.Goni, "A uniform approach to design a federated system using BACK", *Proc. Terminological Logic Users Workshop*, Berlin, December 1991.

[24] Immerman, N., "Upper and lower bounds for first-order expressibility", *J. Comp. Syst. Sciences*, 25, 1982, pp. 76-98.

[25] Kolaitis, P., M. Vardi, "On the expressive power of Datalog: tools and a case study", *Proc. ACM PODS'91*, pp. 61–71.

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

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

[28] van der Meyden, R., *The complexity of querying indefinite information*, PhD Thesis, Rutgers University, October 1992. (The portion relevant to this paper to appear in *Theoretical Computer Science*)

[29] Nebel, B., "Computational Complexity of Terminological Reasoning in BACK," *Artificial Intelligence*, 34(3):371–383, April, 1988.

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

[31] Nebel, B., "Terminological cycles: semantics and computational properties", in J. Sowa, editor, *Principles of Semantic Networks*, Morgan Kaufmann, 1991.

[32] Nebel, B. and C. Peltason, "Terminological Reasoning and Information Management", D. Karagianis editor, *Information Systems and Artificial Intelligence.*

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

[34] 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.

[35] Sagiv, Y. and M. Yannakakis, "Equivalence among relational expressions with the union and difference operators", *JACM 29:1*, 1981, pp.633-655.

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

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

[38] Schmolze, J.G., D. Israel, "KL-ONE: semantics and classification", in *Research in Knowledge Represetnation for Natural Language Understanding — ANnual Report*, Tech Report 5421, BBN Laboratories, 1983.

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

[40] J. Ullman, *Principles of database and Knowledge-Base Systems*, Computer Science Press, 1989.

[41] 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.