Asking Queries about Frames

Alexander Borgida and Deborah L. McGuinness*

Dept. of Computer Science
Rutgers University
New Brunswick, NJ
{borgida,dlm}@cs.rutgers.edu

Abstract

Frame-based knowledge representation and reasoning systems typically provide procedural interfaces for asking about properties of individuals and concepts. We propose an alternative declarative approach that extends standard interface functionality by supporting selective viewing of components of complex objects. Instead of just returning sets of individuals, our queries match concepts and *filtered* fragments of descriptions. The query language is an extended form of the language used to describe the knowledge-base contents, thus facilitating user training.

In this paper, we describe a variety of possible semantics for answering queries in description logics. We investigate the algorithms required when answers are deduced by matching queries against a "structural normal form" of descriptions. As part of our approach, we introduce a useful refinement of the notion of structural subsumption.

1 Motivation

Our experience developing large knowledge bases using frame-based knowledge representation and reasoning systems (FR-KRSs) — CLASSIC [3] in our case — has revealed a significant lacuna: when displaying or explaining complex concepts and individuals, the system's built-in functions (such as print) flood the knowledge engineer with uninteresting and irrelevant information. For example, these functions do not differentiate widely-known "definitional" knowledge (e.g., PEOPLE's ages should be of type INTEGER) from more specific constraints (the age value is restricted to be of type "[min 14]" – the set of integer values no less than 14). Also they can not distinguish

---

*AT&T Labs-Research, Murray Hill, NJ 07974, dlm@research.att.com

special-purpose roles/slots (e.g., roles describing how to retrieve information from data bases or how to produce multimedia displays), which are of interest for different audiences. Moreover, knowledge representation and reasoning systems (KRSs) typically focus on context independent needs and neglect context dependent, domain specific requirements. Still, many applications demand attention to context and the domain. For example, although in general we might not care to see the sugarContent of an instance f of the FOOD class, this becomes crucial when f appears as filler for the eats role of a diabetic person. Currently, such difficulties can be resolved only by writing one-of-a-kind special-purpose procedures, which use the application program interface (API) to the KRS. For example, one can retrieve the value restriction on the age role by invoking get-value-restriction(Joe,age); and then check whether the answer returned is strictly more specialized than the concept INTEGER, by using the function concept-subsumes.

This is symptomatic of a more general problem with KRSs in general, and Description Logics (DLs) in particular: Considerable effort has been expended in looking for "good" languages in which to make assertions to be stored in the knowledge base (for so-called TELL operations) — languages which, for example, permit only statements whose logical consequences are effectively computable. In contrast, very little attention has been paid to languages for asking questions (or for describing answers), leaving this aspect to the procedural interface. This reliance on a procedural solution, rather than a declarative solution, is undesirable for several reasons:

1. *Declarative* queries are preferable since they are more concise, can be analyzed (e.g., for coherence), can be optimized, etc. The advantages of powerful declarative query languages have in fact long been recognized and exploited in database systems.

2. Declarative queries can also be cached, organized and reused more easily, which is increasingly important for tasks such as data exploration and op-

timization [4, 5].

3. A query notation unrelated to either the TELL language or to the structure of the objects is harder for users to learn.

4. Even powerful query languages based on First Order Predicate Calculus (FOPC), such as in LOOM[13], do not to support questions returning concepts and formulas, such as the ones provided by the API procedures.

We will propose a declarative language for querying FR-KRSs that addresses many of the above issues. In particular, queries will resemble concepts in a natural way, and answers to queries can be concepts, or even new descriptions in DLs, not just individual values. We will show with examples that the precise notion of "desired answer" is in fact not obvious, and will choose to pursue our own research in the paradigm of concepts in "structural subsumption" normal-form. In order to develop and study query-answering algorithms, we will need to take a detour to refine the notion of structural subsumption introduced in [6]. Finally, we will describe some enhancements to the query language which have been motivated by our initial practical applications [19, 4].

An earlier design, which lead in part to the ideas in this paper, has been implemented for the CLASSIC system [18] and has been used in an application to greatly simplify the presentations and explanations of configurations [15].

2 Concepts and Queries

Frame-based knowledge base management systems represent information using individuals, grouped into concepts, and inter-related by binary relationships called slots or roles. Concepts are related by a subsumption/IsA relationship, written here as $\implies$, while individuals are instances of concepts, written as $\rightarrow$. Thus, we will write PERSON $\implies$ MAMMAL and Charles $\rightarrow$PERSON.

2.1 Description Logics

For concreteness, we shall use the syntax of a DL resembling CLASSIC, which supports both primitive concepts, such as BEAR, and composite ones. Structured, composite concepts are built with so-called concept constructors, and may be, among others, enumerations of individuals (e.g., [one-of Amy Lulu]), or number and type restrictions on roles. For example, the concept PICKY-EATER, defined as [and [at-most 2 eats] [all eats SWEET-FOOD]], is intended to denote individuals that have no more than two fillers for the eats role, and all such fillers must be instances of the concept SWEET-FOOD, defined elsewhere.
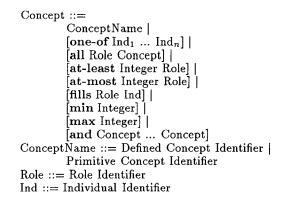
Concept ::=
 ConceptName |
 [one-of Ind$_1$ ... Ind$_n$] |
 [all Role Concept] |
 [at-least Integer Role] |
 [at-most Integer Role] |
 [fills Role Ind] |
 [min Integer] |
 [max Integer] |
 [and Concept ... Concept]
ConceptName ::= Defined Concept Identifier |
 Primitive Concept Identifier
Role ::= Role Identifier
Ind ::= Individual Identifier

Table 1: Syntax of MiniDL

CONCEPT	DENOTATION
primitive A	$A^{\mathcal{I}}$
[one-of B$_1$,...,B$_m$]	$\{B_1^{\mathcal{I}}, \ldots, B_m^{\mathcal{I}}\}$
[min n]	$\{d \in \mathcal{N} \mid d \geq n\}$
[max n]	$\{d \in \mathcal{N} \mid d \leq n\}$
[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]	$\{d \in \Delta^{\mathcal{I}} \mid |p^{\mathcal{I}}(d)| \geq n\}$
[at-most n p]	$\{d \in \Delta^{\mathcal{I}} \mid |p^{\mathcal{I}}(d)| \leq n\}$
[fills p B]	$\{d \in \Delta^{\mathcal{I}} \mid B^{\mathcal{I}} \in p^{\mathcal{I}}(d)\}$
[and C D]	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$

Table 2: Semantics of MiniDL

The syntax of the language is given by the grammar in Table 1. In Table 2, we provide the denotational semantics of this DL fragment, using the notion of a valuation $\mathcal{I}$ that assigns a subset of a domain $\Delta^{\mathcal{I}}$ to primitive concepts, and assigns a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to primitive roles.

Finally, a particular KB contains, in addition to concept declarations, information about the membership of individuals in concepts, and their role fillers: "Amy is a BEAR with at least 3 friends", "Lulu's age is 9". Both of these can be recorded as membership assertions: Amy→[and BEAR [at-least 3 friends]], Lulu→[fills age 9].

Systems based on DLs perform a number of operations on descriptions and individuals, such as checking for consistency and "classifying" concepts into a subsumption or IsA hierarchy, where D1 subsumes D2 iff the denotation of D1 contains that of D2 in all possible interpretations. Thus, MAMMAL subsumes PERSON if and only if every individual that is a PERSON is also a MAMMAL in all possible interpretations.

2.2 Query Concepts

The conventional knowledge-level view of asking a knowledge base KB some question is to determine whether the query formula Q is logically entailed by it: $KB \models Q$. While this yields only "Yes" or "No" as answers, a standard generalization for first-order logic KBs is to ask queries with free variables, such as $Q(?x, ?y)$; the answer is the set of valid substitutions for the variables: { σ | σ maps $FREE\text{-}VARS(Q)$ to constants such that $KB \models \sigma(Q)$ }, where $\sigma(Q)$ stands for Q with every variable y replaced by $\sigma(y)$. For example, given the FOPC KB { likes(Ann,Deb), likes(Ann,Honey) }, the query likes(Ann,?x), returns as answers the substitutions mapping ?x to Deb and Honey respectively (denoted by { ?x↦Deb } and { ?x↦Honey }).

This suggests that we introduce variables (here, syntactically distinguished by a preceding question mark) for frame questions as well. To achieve our goal of querying portions of frames or descriptions, we allow variables to appear anywhere identifiers or constants may appear in the syntax of concepts (viz. Table 1). For example, [at-least ?n friends] or [all friends [fills liveAt ?x]] will be query concepts. Such a query Q will be directed against a particular individual i (written as i::Q) or concept D (written as D::Q), so that the process of answering D::Q will resemble the notion of *matching*. When such a query concept is matched against an individual or concept, the result will be a (set of) substitution(s), or *failure*, in case no substitution can be found.

The following examples are intended to provide an initial, intuitive feeling for the notion of query answering. Consider the knowledge base KB1= {
   Lulu→[and BEAR [all friends NICE]],
   Lulu→[fills age 9],
   Lulu→[at-least 3 friends] }.
The match Lulu:[fills age ?x] should naturally return the substitution ?x↦9, while the match Lulu:[all friends ?X] returns ?X↦NICE.[1] More generally, one can have complex query concepts, such as
   [and [at-least ?n friends]
      [all friends [all age ?a]]]
which, when matched against Lulu, would result in failure, since the nested match NICE::[all age ?a] does not produce a substitution.

Clearly, this approach provides two of the advantages we desired: the query language mirrors the natural form of the knowledge base contents, and it allows queries that return (portions of) frames, as in [all friend ?X]. We do however need to be circumspect

---
[1]Note that variables are typed by context as matching individuals, roles, concepts, integers, or sets thereof. To facilitate reading, we use upper case variables for concepts and lower case for everything else in our examples.

since matching is more complicated than it appears at first glance. We will proceed by first presenting a number of less obvious examples, and then examining in greater detail two approaches based on structural subsumption. We will return in the end to describe some enhancements to the query language.

In the rest of this paper we will concentrate on the notion of matching query concepts against concept descriptions, rather than individuals, both because of lack of space and because in several systems reasoning about individuals can be reduced to reasoning about concepts. (We have investigated matching query concepts against individuals in a slightly different setting in [14].)

2.3 On the semantics of matching

First, note that it would be very reasonable to expect the match [all pets NOTHING] :: [at-most ?n pets] to succeed with ?n↦0. The meaning of NOTHING is the empty set, thus [at-most 0 pets] is semantically equivalent to [all pets NOTHING]. Suppose we treat descriptions as terms, with concept constructors as functors, so that [all pets NOTHING] is represented as all(pets,NOTHING). A *purely syntactic* notion of matching is then clearly undesirable, since the standard unification of the terms all(pets,NOTHING) and at-most(?n,pets) fails.

Even if we used unification modulo a theory of concept equivalence, the result would not be satisfactory, because the match of PICKY-EATER, whose definition was [and [at-most 2 eats] [all eats SWEET-FOOD]], against [all eats ?X] would fail, because the conjunct with the at-most restriction is missing in the pattern. This match failure would prevent us from using queries to extract interesting *parts* of concepts, as we originally wanted.

We are therefore naturally led to consider the meaning of C::Q as a request for a substitution σ such that C is *subsumed by* $\sigma(Q)$. For example, given the concept PICKY-EATER above and the query concept [all eats ?X], we can would like the substitution ?X↦SWEET-FOOD since this substitution applied to the query concept yields [all eats SWEET-FOOD], which subsumes PICKY-EATER. However, even here we run into trouble because the match PICKY-EATER::[at-most ?n eats] would return in addition to ?n↦2, an infinite number of substitutions, ?n↦3, ?n↦4, etc., since [at-most 3 eats], etc., all subsume PICKY-EATER. Yet, again, for the purpose of displaying part of PICKY-EATER, we only want to see the first substitution. The reason we would prefer the first is because of all the substitutions in the infinite set, this one, when applied to the query concept, results in the most specific concept. Therefore, we propose the following as a baseline definition:

Definition 1 *Given query concept Q and description D, the match D::Q succeeds with substitution σ if and only if D is subsumed by σ(Q), and there is no σ' such that D is subsumed by σ'(Q), with σ'(Q) strictly subsumed by σ(Q).*

Let us review now the possible results of matching a query concept against a description.

Obviously, the match may fail because there is no substitution σ such that σ(Q) subsumes D. But it may also fail according to the above definition because there is no such *minimal* substitution. The simplest example of this case involves individual matching: if we have the KB describing Amy as a bear with exactly one best friend, namely herself, { Amy:BEAR, Amy→[fills bestFriend Amy], Amy→[at-most 1 bestFriend] }, then it follows that Amy is an instance of the following sequence of descriptions: [all bestFriend BEAR], [all bestFriend [and BEAR [all bestFriend BEAR]]], etc., each of which is subsumed by the previous one. Therefore, for the query [all bestFriend ?X], a substitution σ mapping ?X to any one of the above descriptions provides a situation where Amy is an instance of σ[all bestFriend ?X], but there is no most general lower-bound to this infinite descending chain.

If we want to consider only concept subsumption, the same problem arises in languages supporting role-chain equality (same-as), where the conjunction of [all bestFriend BEAR] and [same-as (bestFriend) (bestFriend bestFriend)]] is subsumed by every description in the above sequence.

When the match succeeds (still using Definition 1), there are once again a number of possibilities. It may succeed with a single substitution, as in the case of matches for query concepts like [all p ?X] or [at-least ?n p]. Or it may succeed with multiple substitutions, as in the case when we match [fills friend ?x] against [and [fills friend Amy] [fills friend Lulu]], where ?x↦Amy and ?x↦Lulu both yield valid matches. In fact, there may be circumstances where there are an infinite number of minimal substitutions, as in the case of matching [and [fraction-min ?n] [fraction-max ?n]] against [and [fraction-min $\frac{1}{2}$] [fraction-max $\frac{2}{3}$]], which yields as answers substitutions for ?n of all rational numbers between a half and two thirds.

In all these cases the multiple substitutions returned by the match seem equally acceptable. However, consider the match
    [and BEAR YOUNG] :: [and BEAR ?X].
In this case, ?X↦YOUNG and ?X↦[and BEAR YOUNG] are both valid substitutions according to our definition. In fact, so is ?X↦[or YOUNG C] for any concept C disjoint from BEAR, such as WOLF. The latter substitu-

tions seem undesirable, which would seem to indicate the need for further restrictions on the substitutions to be returned by a match; in this case, one might require that each variable be assigned as specific a value as possible. However, the following example then becomes problematic: matching
    [and [all width [min 3]]
        [all length [max 4]]]
against
    [and [all width [min ?n]]
        [all length [max ?n]]]
would normally return substitutions 3 or 4 for ?n. Yet this last definition would quite arbitrarily pick the "smallest" value of the two, say 3 in this case.

Rather than further explore the intricacies of an abstract specification of matching, we will turn to a theory of normal forms for DLs. This will yield both a particular approach to dealing with the issues above, and algorithms for computing the matching substitutions. Note that several other new operations on DLs, such as least common subsumer [6] and concept difference [16], also required restriction to normal-form in order to yield sensible results. However, this does not preclude the possibility of future work on query matching in a more general setting.

3 Structural subsumption with normalized descriptions

In looking for a suitable theory of normal forms, we note that the three DL systems widely used in practice (BACK, CLASSIC, LOOM) are implemented by first preprocessing each concept so that later operations are made easy. This initial normalization phase detects inconsistencies, and makes explicit all the implied facts or descriptions by constructing a *normal form* containing the most specific forms of the different *kinds* of descriptions entailed. For example, the normal form of the description [at-most 0 p] might be and({ at-most(0,p), all(p,NOTHING) }). For subsumption, it is these specific subterms that are used in a relatively straightforward procedure, which we call "structural subsumption".

Our version of structural subsumption (based on our earlier work on least common subsumers [6, 2]) proposes that every concept constructor **k** is to be viewed as a term constructor with a single argument, which is a value from some set D partially ordered by the relationship $\preceq_D$ (written as $D^{\preceq_D}$). *Structural subsumption checking* has the property that
$$\mathbf{k}(\alpha) \Longrightarrow \mathbf{k}'(\beta) \text{ is true iff } \mathbf{k} = \mathbf{k}' \text{ and } \alpha \preceq_D \beta.$$

The significance of this property is that since the normal form of a concept is usually a (conjunctively interpreted) set of descriptions built with various other constructors, one can restrict oneself to comparing ele-

ments built with the same constructor, without considering interactions between different kinds of constructors. The purpose of the normal form is then to allow subsumption to be computed using only structural comparisons of the above form. To assist this task, normal forms are also required to be non-redundant (e.g., in a set of values, none subsumes another).

Let us consider some examples of constructors and their normal form supporting structural subsumption, while at the same time introducing some refinements.

Our **min** concept constructor takes an integer as an argument, with the ordering being $\geq$, since $\mathbf{min}(n) \implies \mathbf{min}(m)$ iff $n \geq m$. On the other hand, **max** also takes a integer as an argument, but the ordering is $\leq$. Since $\leq$ and $\geq$ are in fact inverses over the domain $\mathcal{N}$ of numbers, we will find it essential later to recognize this explicitly by picking one of them, say $\geq$, to be the standard ordering for $\mathcal{N}$, and use a new, auxiliary term constructor **invert**, to represent explicitly the cases in which the inverted partial order is used. So [min 2] and [max 3] will have normalized form $\mathbf{min}(2)$ and $\mathbf{max}(\mathbf{invert}(3))$, with **min** and **max** being said to have type $\mathcal{N}^{\geq}$ and $Invert(\mathcal{N}^{\geq})$ respectively. Of course, the comparison rule for the **invert** operator will be $\mathbf{invert}(\alpha) \preceq_{invert} \mathbf{invert}(\beta)$ iff $\beta \preceq \alpha$.

Some description constructors, such as **min**, take an argument that is a value from a "basic" partial order, such as integers ordered by $\geq$ ($\mathcal{N}^{\geq}$), individuals ordered by equality ($\mathcal{I}nd^{=}$), primitive concepts and roles ordered by subsumption ($\mathcal{P}rimitiveConcept \overset{\implies}{}, \mathcal{P}rimitiveRole \overset{\implies}{}$). (For convenience, we will henceforth drop the partial order superscript.)

Other constructors take a composite value as an argument. For example, the **one-of** constructor (used to denote enumerated concepts) takes a set of individuals as an argument, while **all** takes 2 arguments: the role name and the concept. Since, we want each concept constructor **k** to take a single value as an argument, we shall extend the term notation used for concepts to also describe composite values by introducing the **set** and **tuple** functors. Thus, the normal form of [all p C] will be **all(tuple(p,C))**, while the normal form of [one-of 2 4 6] will be **one-of(set({2,4,6}))**.

We must then extend the ordering $\preceq$ to composite domains. In $Tuple(D_1^{\preceq_1}, D_2^{\preceq_2})$, the values from $D_1 \times D_2$ are ordered by $\preceq_{tuple}$ according to componentwise comparison: $(n_1, r_1) \preceq_{tuple} (n_2, r_2)$ if and only if $n_1 \preceq_1 n_2$ and $r_1 \preceq_2 r_2$. For example, the intuitively valid subsumption [all children FRENCH] $\implies$ [all sons EUROPEAN] shows that the **all** constructor is contravariant on the role argument, and hence its normalized type is $Tuple(Invert(\mathcal{P}rimitiveRole), Concept)$. As a result, we get the proof

[all children FRENCH]

CONCEPT	Type of Normalized Arg.
primitive A	$Setof(\mathcal{P}rimitiveConcept)$
one-of $B_1,...,B_m$]	$Invert(Setof(\mathcal{I}nd))$
[min n]	$\mathcal{N}$
[max n]	$Invert(\mathcal{N})$
[all p C]	$Tuple(Invert(\mathcal{R}ole), Concept)$
[some p C]	$Tuple(\mathcal{R}ole, Concept)$
[at-least n p]	$Tuple(\mathcal{N}, \mathcal{R}ole)$
[at-most n p]	$Tuple(Invert(\mathcal{N}), Invert(\mathcal{R}ole))$
[fills p B]	$Tuple(\mathcal{R}ole, Setof(\mathcal{I}nd))$
[and C D]	$Setof($ non-redundant terms built with the above constructors$)$

Table 3: Normal Form for MiniDL

$$\implies \text{[all sons EUROPEAN]}$$
$$\text{iff}$$
$$\text{all(tuple(invert(children),FRENCH))}$$
$$\implies \text{all(tuple(invert(sons),EUROPEAN))}$$
$$\text{iff}$$
$$\text{tuple(invert(children),FRENCH)}$$
$$\preceq \text{tuple(invert(sons),EUROPEAN)}$$
$$\text{iff}$$
$$\text{invert(children)} \preceq \text{invert(sons)},$$
$$\text{and FRENCH} \preceq \text{EUROPEAN}$$
$$\text{iff}$$
$$\text{sons} \implies \text{children, and FRENCH} \implies \text{EUROPEAN}$$

which is true, as desired.

For sets, the natural ordering is either $\subseteq$ or $\supseteq$, with one obtainable from the other by inversion: $A \subseteq B$ iff $\mathbf{invert}(A) \supseteq \mathbf{invert}(B)$. For reasons to become apparent later, it is convenient to choose $\supseteq$ as the basic ordering on sets. Therefore, since we expect [one-of 2 4] $\implies$ [one-of 2 4 6], the concept constructor **one-of** should take a value of type $Invert(Setof(\mathcal{I}nd))$ as its argument.

The argument of the **and** constructor is a set of descriptions, and $\supseteq$ gives exactly the right formulation for **and(set({CAT,BROWN}))** to be subsumed by **and(set({CAT}))**. However, the elements of a set can be values that are themselves ordered (e.g., descriptions), and this has to be taken into account during subsumption checking for **set**. In particular, for **and**, we also expect **and(set({CAT,BROWN}))** to be subsumed by **and(set({FELINE}))**. Therefore, in general we need to define the $\preceq_{set}$ relationship for two sets V and W as V$\preceq_{set}$W iff for every element x of W there is an element y in V such that $y \preceq_D x$.

Table 3 summarizes the type of the normalized domains associated with each of the concept constructors in our example DL.

The following "structural $\preceq$" algorithm embodies the above ideas; it computes structural subsumption for descriptions, as well as the $\preceq_{set}$, $\preceq_{tuple}$ and $\preceq_{invert}$ relationships, relying on the $\preceq_B$ functions provided by basic types B:

```
function structural⪯(α, β) {
        if  α and β are  values in  basic type B
            (i.e., N,Ind,PrimitiveConcept,...) with
                partial order compare_B
            then compare_B(α,β)
        elseif α=k(α') & β=k(β') where k is a
                concept constructor
            then structural⪯(α',β')
        elseif α=invert(α') & β=invert(β')
            then structural⪯(β',α')
        elseif α=tuple(α1,α2) & β=tuple(β1,β2)
            then structural⪯(α1,β1) & structural⪯(α2,β2)
        elseif α=set(α') & β=set(β')
            then for every e in β'  choose f in α'
                                where structural⪯(f,e)
        else fail }
```

In the above, **choose x in S where P** is a non-deterministic operation, which fails if there is no value in S satisfying property P.

Note that for some DLs there may be no normal form yielding structural subsumption that is sound and complete with respect to semantic subsumption, or such a normal form may be very expensive to compute. For this reason, sometimes the structural subsumption relationship is incomplete with respect to the semantic subsumption relationship.

We will henceforth consider the case when both the concepts and queries are in normal form, and in fact define subsumption to be the relation computed between descriptions by structural⪯. Therefore structural⪯ computes D::Q in the case where Q has no variables.

4 Direct matching with normal forms

Having defined structural subsumption for descriptions and the concomitant normal form, we are now ready to consider the extensions necessary to match query concepts that have variables occurring in them. We obtain the match(α,β) procedure to be used for this task from the structural subsumption procedure structural⪯(α,β) by making several changes.

First, if one of the arguments of match is a variable, then it is bound to the other argument to yield a substitution.

Second, we need to combine the substitutions returned by the recursive calls. For this, we shall use a function combine, which in this initial simple case requires that the substitutions agree exactly on all variables on which both are defined. Thus, combine($\sigma1,\sigma2$) signals failure if there is some variable ?x such that $\sigma1(?x) \neq \sigma2(?x)$, and otherwise returns $\sigma1 \cup \sigma2$.[2]

---

[2]We consider a substitution as a partial function from variables to values, which can then be viewed as a set of

One additional complication arises because, according to Definition 1, we need to return the most specific substitution realizing a match. To illustrate this point, consider the concept constructor **some**, which intuitively provides typed existential quantification: [some friend TALL] denotes objects that have at least one filler for the friend role that is an instance of TALL. Consider the case of matching the query Q=[some friend [at-least ?n cars]] against the conjunction of [some friend [and TALL [at-least 3 cars]]] and [some friend [and SHORT [at-least 5 cars]]]. Although both the substitutions σ_1:?n↦5 and σ_2:?n↦3 have the property that $\sigma(Q)$ subsumes the concept to be matched, only σ_1 should be returned by match because [some friend [at-least 5 cars]] is itself strictly subsumed by [some friend [at-least 3 cars]].

This difficulty needs to be handled in the matching of sets: Originally, if set(α') is to be matched by set(β'), we would need to find for every e in β' some f in α' so that e matches f; however, if e contains variables, we need to make sure that e matches one of the elements of α which yields *the most specific* result. Similarly, if it is f that has variables (because of an intervening occurrence of **invert**), the match needs to return the most general possible value, so that the inversion will yield the most specific value later.[3]

As a result, we get the following algorithm for matching query concepts against descriptions:

```
function match(α, β)  {
    ;; returns a matching substitution or signals failure
    if (β is a variable)
        then return substitution { β↦α }
    elseif (α is a variable)
        then return substitution { α↦β }
    elseif α and β are  values in  basic type B
        (i.e., N,Ind,PrimitiveConcept,...) with partial
            order compare_B
        then if compare_B(α,β)
                then return empty substitution
                else fail
    elseif α=k(α') & β=k(β') where k is a constructor
        then match(α',β')
    elseif α=invert(α') & β=invert(β')
        then match(β',α')
    elseif α=tuple(α1,α2) & β=tuple(β1,β2)
        then return combine(match(α1,β1),match(α2,β2))
    elseif α=set(α') & β=set(β')
        then{result := empty substitution;
            for every e in β' {
                choose f in α' where σ:=match(f,e)
```

2-tuples.

[3]Note, that only one argument of match may have variables, since originally the first parameter must be a (variable-free) concept, and invert only swaps the two arguments of match.

```
              succeeds and σ(e) is minimal
              and σ(f) is maximal;
      result := combine(result,σ)}
        return result; }
   else fail }
```

The algorithm just presented is sound, in the sense that

Theorem 1 *If* match*(D,Q) returns substitution σ, then σ is a substitution that is a witness that D::Q succeeds according to Definition 1.*

Note that the match procedure collapses to structural subsumption checking when the second argument has no variables; in this case match(α,β) returns the empty substitution if β subsumes α, and fails otherwise. Also, in order to guarantee the minimality of the substitution, the proof of this theorem requires that the normal form of both the concept and the query be non-redundant, in the sense that in any set, no distinct pair of elements match each other.

As we shall see later, the algorithm is not complete. Moreover, the non-determinism present in the choose operation, and the possible relationships between choices at difference points, can cause complexity problems:

Theorem 2 *Consider the DL with concept constructors* and, all, *and primitive concepts. The problem of determining whether* match*(D,Q) succeeds in this language is NP-complete. In contrast, both subsumption and structural subsumption are polynomial time.*

The proof encodes propositional satisfiability into the choice of matches for role variables and primitive concept variables.

The following theorem indicates several cases where problems do not arise:

Theorem 3 *The complexity of checking that matching succeeds is proportional to the complexity of computing structural subsumption with normalized concepts (1) when no variable occurs more than once in the query; (2) when the* choose *operation always yields at most one successful result.*

The first case is relevant since it corresponds to what is known as "matching" with ordinary terms, which is a linear time operation used in programming languages such as SML.

5   Semantic matching with normal form

In the preceding sections we have defined structural subsumption and the normal form for descriptions that supports it, and then modified this subsumption algorithm to find matches when a query concept is passed to it as an argument. The result resembles matching of standard first-order terms except that we use subsumption instead of identity in comparing constants, and we treat the set construct specially, because we have commutative/associative operators such as conjunction. We now proceed to make matching closer to the ideal specified in Definition 1.

To illustrate the problem, observe that the previous algorithm fails to match [and [all p ?X] [all q ?X]] against [and [all p DOG] [all q CAT]] even though the substitution ?X↦ANIMAL would yield a subsuming concept. The solution is to relax the combination of substitutions: if σ1(?X)=DOG, σ2(?X)=CAT, then combine(σ1,σ2)(?X) should be ANIMAL, or in fact any least common subsumer of CAT and DOG, such as [and MAMMAL HOUSE-PET].

To find such substitutions, note that match(a,?x) succeeds whenever ?x is some value *greater* than a, while match(?x,b) succeeds whenever ?x is bound to some value less than b. Since a variable may appear in several places, there may be multiple restrictions on it that act as lower or upper bounds. Hence our algorithm will associate with a variable a pair (A, B), representing the set of possible values $\{ f \mid a_i \preceq f$ for each $a_i \in A$, $f \preceq b$ for each $b_j \in B \}$ that could be substituted for it.

Therefore the first two statements of match(α,β) are refined as follows:

```
if (β is a variable)
    then return β↦({ α },{ })
elseif (α is a variable)
    then return α↦({ },{ β })
```
As a result, match(**min**(3),**min**(?n)) would call match(3,?n), which would return ?n↦({ 3 },{ }).

In turn, combine is modified to behave as follows: if $\sigma_1(?x) = (A_1, B_1)$ and $\sigma_2(?x) = (A_2, B_2)$ then combine$(\sigma_1,\sigma_2)(?x) = (A_1 \cup A_2, B_1 \cup B_2)$.

Finally, we need to extract the required substitutions from the pair of sets returned by the modified match. If match returned ?x ↦(A,{ }), we an answer substitution that maps ?x to any least upper bound of the set A. (If no such value exists, then failure is signaled.) Similarly, for ?x↦({ },B) we return any greatest lower bound of the set B. If the result of match maps ?x to (A, B), where neither A nor B are empty, then we return all substitutions ?x↦f, where f is greater than any value in A, and less than any value in B.

Therefore, if we match the query concept set({ **min**(?n), **max**(**invert**(?n)) }) against set({ **min**(2), **max**(**invert**(4)) }), we get the answer ?n↦({ 2 }, { 4 }). This permits values between 2 and 4 inclusive to be substituted for ?n. Matching the same pattern against set({ **min**(4), **max**(**invert**(2)) }) fails

because ({ 4 }, { 2 }) is not satisfied by any value.

A simple special case in which the above conditions can be effectively dealt with is when all basic domains are lattices, because in this case one must only keep the least upper bound of A, and the greatest lower bound of B.

Theorem 4 *If all basic domains are either lattices (i.e., least upper bound and greatest lower bound are unique) or sets ordered by the identity relation, then the enhanced* match*(D,Q) succeeds only if* D::Q *according to Definition 1, when* C *and* Q *are normalized.*

6 Enhancements

Our experience with using query concepts shows that the query language needs to be more expressive than the language of stating facts and definitions in the knowledge base. We briefly sketch several extensions that we have found useful. (For more details, see [14].)

First, we allow restrictions on variable bindings via a nested as clause. For example
        Sally:[at-least ?n [as [max 4]] friends]
will match only if Sally's lower-bound on the number of friends is less than four. This is equivalent to a conjunction of matches
        Sally:[at-least ?n friends] & ?n:[max 4]
but is syntactically preferable since it gives the query a structure resembling that of the knowledge base itself. As another example, the following query concept
        [and [fills friends ?x
                        as [and BEAR [fills age ?y]]]
                [all friends ?Z as SMALL]]
when matched against an individual, say Sally looks for friends of Sally who are known to be BEARs, and also asks for their ages. It also retrieves the most specific restriction on the friends slot which is itself, subsumed by SMALL. We found this kind of nested as clause particularly useful when printing or explaining aspects of role fillers. For example when generating a parts list for a stereo system configuration in [15], we wanted to obtain the price information on the individual stereo components. The simple pattern that should be matched against stereo systems is:
        [fills component ?x as [fills price ?y]].

As illustrated above, we allow a qualifying query Q to apply to variables that matched a concept, as in [all friends ?Z as Q], and, to be consistent, the value bound to ?Z should be viewed as an individual instance of some (meta)class described by Q. For this reason, and following a standard object-centered policy, we will think of descriptions as meta-individuals in their own right, which have their own roles and may belong to their own meta-classes. Among others, we will posit for each meta-individual a number of attributes concerning the IsA hierarchy, such as strictlySubsumedBy , immedi-

atelySubsumedBy , etc., as well as meta-classes such as INTERESTING-CONCEPTS, DATABASE-ACCESS-ROLES, and NAMED-DESCRIPTIONS. In this way, if we want to see the value-restriction on the eats role only if it is more specific than the generic FOOD concept, we match against the query [all eats ?C as [fills strictly-SubsumedBy FOOD]].

The above mechanism can of course be used to retrieve aspects of the IsA hierarchy. For example, the "parent concepts" of class PICKY can be obtained by the query
        PICKY :: [fills immediatelySubsumedBy ?X].

Following [7, 17], a different way of enhancing the query language is to use a richer set of concept and role constructors. Among others, in those FR-KRSs that make the open-world assumption, it is possible for the system to know that Sally has at least three friends, while only knowing the identity of two of them. For this reason, we include epistemic constructs [9] in the language. (A simple way to do this is to include an epistemic version of each constructor: [at-least-k ?n p] would query how many fillers are currently known for p. Our method, discussed in [14], includes **known** as a role constructor and then extends the grammar appropriately.) In addition, as a result of our experiences with CLASSIC, we believe it is imperative to include some form of extensibility in the language. For this reason we propose a **testMatch** constructor, which takes as arguments a function for matching and a list of variables to bind: [testMatch compute-and-bind-oldest-age-of-friends '(?x)].

7 Related Work

We were inspired to use variables ?x in a language dealing with description logics by work in databases, as well as LOOM's FOL assertion/query language, and by other papers [8, 11] that have explored the combination of variables and frames in the context of Horn logic. As mentioned earlier, Lenzerini et al [7, 9] considered the problem of querying DL KBs using other concepts as queries, and showed various ways of using more expressive concept languages for asking questions. However, in all these cases variables were only ranging over individuals, with roles and concepts being used as unary and binary predicates. It was thus not possible to query parts of the frames themselves, which was the most significant aspect of our initial motivation — to reduce the amount of material being displayed.

In the database area, object-oriented query languages such as XSQL [12] have notation which facilitates the querying of path-expressions and allows variables for attributes as well as values: PERSON X WHERE X.Residence[Y].City['New York'] AND X.R.City[W]. Our work extends this through the use of subsumption as part of query matching, and the much more general nature of

frames.

Finally, our earlier work on least common subsumers for DLs [6] also used the notion of structural subsumption. The present work considerably elaborates the theory of normal forms presented in that paper. The alternative theory of normal forms for structural subsumption introduced in [16] might also have provided a good foundation for concept matching. Unfortunately, that theory relies on the notion of "clausal description" — one which has the property that if A≡and(A',B) then either B is A or B is the universal concept. This does not work for the one-of constructor of CLASSIC, since [one-of 1 2]= [and [one-of 1 2 3] [one-of 1 2 4]] , thus, there would be no normal form for one-of.

8 Desiderata Revisited

We have introduced a declarative language for inquiring about information in FR-KRSs which support notions such as roles or slots, inheritance, subsumption, etc. As desired, the query language is derived from the standard language for telling information to the KB, mostly by introducing variables for all user-specified identifiers. This query (or ASK) language can be used to "decompose" a description or frame into its various components in a uniform, declarative manner rather than using a long list of procedures. For example, PERSON::[all friends ?X] accesses the all-restriction on the friends role of PERSON, in a way that was formerly possible only using function calls such as get-all-restriction(PERSON,friends).

Returning to our initial motivation, query concepts can be used to set up "masks" that work as filters and provide only the interesting information for display. In our implemented system this is done by attaching a set of query concepts to a frame[4] so that only portions matched by *some* query concept are shown or explained. Thus, we can avoid obvious information about what restrictions apply to livingParents by using the query concepts [all livingParents ?X as [fills strictlySubsumedBy PERSON]] and [at-most ?n [as [max 1]] livingParents]; the former succeeds only if the type constraint on livingParents is more specialized than PERSON, while the latter succeeds only if there are fewer than 2 parents.

Similarly, although the query concept describing the aspects to be printed for FOOD would not have a part matching the role sugarContent, we can require to see this for diabetics by associating [all eats [fills sugarContent ?x]] with the class DIABETIC.

We claim that the approach is applicable in general to

---

[4]There is inheritance and combination of such query concept filters. Also, different query concepts may be used for explanation than those used for printing objects.

FR-KRSs. For example, the KRL [1] frame specified as

```
PATIENT  unit specialization
   <self (a PERSON )>
   <treatedBy (a DOCTOR with locations= (item NJ))>
```
can be queried by

```
< treatedBy ( a ?X  with ?r = ( item  ?v))>
```
Additionally, a KEE [10] frame of the form

```
PATIENT with slot treatedBy
  [ValueClass=DOCTOR,
   Cardinality.Min=2,
   Values=Kildare]
```
would be queried by

```
slot treatedBy [ValueClass=?X].
```

We have presented the initial exploratory steps concerning the notion of query concepts. Several other significant questions need to be addressed, including the treatment of the same-as construct in structural subsumption, the normalization of query concepts themselves, and the process of answering queries in DLs that do not use structural subsumption. One intriguing idea is to think of a query concept Q as a meta-concept with denotation $\mathcal{D}(Q)=\{\ D \mid \text{match}(D,Q) \text{ succeeds } \}$. This permits us to organize query concepts in a subsumption hierarchy defined as $Q1 \Longrightarrow Q2$ iff $\mathcal{D}(Q1) \subseteq \mathcal{D}(Q2)$. One reason this is interesting is because the subsumption hierarchy of ordinary queries has been found useful in the past (e.g., [5]). Moreover, this notion may be related to matches of queries against queries, $Q1::Q2$, which resembles the notion of unification extended with subsumption reasoning.

In conclusion, in addition to having proposed and successfully used a mechanism of clear benefit to FR-KRSs, we hope that the motivation and subtleties of query concept semantics and matching presented in this paper may lead to new directions of research for DLs; among others, we point out that query concepts are the natural counterparts of polymorphic types in programming languages, which have been extensively studied.

Acknowledgment

References

[1] D. Bobrow, T. Winograd, "KRL: another perspective", *Cognitive Science*, 3(1), pp.29-42, 1979.

[2] A.Borgida, 'Structural Subsumption: What is it and why is it important?', *1992 AAAI Fall Symposium: Issues in Description Logics*, pp.14-18.

[3] A. Borgida, R. J. Brachman, D.L McGuinness, and L. Alperin Resnick. CLASSIC: A Structural Data Model for Objects. In *Proc. SIGMOD'89*, Portland, Oregon, June 1989, pp. 59–67.

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

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

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

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

[8] Donini, F., Lenzerini, M., Nardi, D., Schaerf, A. "A hybrid system integrating Datalog and concept languages", *AAAI Fall Symp. on Principles of Hybrid Reasoning*, 1991.

[9] Donini, F., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A., 'Queries, Rules and Definitions as Epistemic Sentences in Concept Languages', *Foundations of Knowledge Representation and Reasoning*, Springer LNAI 810, pp 111-132, 1994.

[10] R. Fikes, T. Kehler, "The role of frame-based representation in reasoning", *CACM* 28(9), September 1985, pp.904–920.

[11] Levy, A., Rousset., M.-C., 'CARIN: A Representation Language Combining Horn Rules and Description Logics', *Proceedings of the International Workshop on Description Logics - DL-95*, pp 44-51, Roma, Italy, 1995

[12] M. Kifer, Won Kim, Y. Sagiv: Querying Object-Oriented Databases.*Proc. SIGMOD'92*, pp.393-402.

[13] R. M. MacGregor and R. Bates. The LOOM Knowledge Representation Language. Technical Report ISI/RS-87-188, USC/Information Sciences Institute, Marina del Ray, CA, 1987.

[14] D. L. McGuinness. Explanation in Description Logics. Rutgers University, PhD thesis, 1996.

[15] D. L. McGuinness, L. Alperin Resnick, and C. Isbell. Description Logic in Practice: A CLASSIC Application. In *Proc. IJCAI'95*, Montreal, 1995.

[16] G. Teege, "Making the Difference: A Subtraction Operation for Description Logics." *Proc. KR-94*, Bonn, Germany, (May, 1994), pp.540-550.

[17] Patel-Schneider, P. F., Brachman, R. J., and Levesque, H. J., "ARGON: Knowledge representation meets information retrieval", in: *Proceedings First Conference on Artificial Intelligence Applications*, Denver, Colorado (1984) 280–286.

[18] L. Alperin Resnick, A. Borgida, R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, C. Isbell, and K.Zalondek. CLASSIC description and reference manual for the Common Lisp implementation: Version 2.3. AI Principles Research Department, AT&T Bell Laboratories. 1995.

[19] Wright, J. R., Weixelbaum, E. S., Brown, K., Vesonder, G. T., Palmer, S. R., Berman, J. I., Moore, H. H., A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T Network Systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, pp.183–193, 1993. A version of this paper also appears in *AI Magazine*, 1993, pp. 69-80.