# An Assertion Retrieval Algebra for Object Queries over Knowledge Bases

**Jeffrey Pound, David Toman, Grant Weddell, and Jiewen Wu**
Cheriton School of Computer Science, University of Waterloo, Canada
{jpound, david, gweddell, j55wu}@uwaterloo.ca

## Abstract

We consider a generalization of instance retrieval over knowledge bases that provides users with assertions in which descriptions of qualifying objects are given in addition to their identifiers. Notably, this involves a transfer of basic database paradigms involving caching and query rewriting in the context of an assertion retrieval algebra. We present an optimization framework for this algebra, with a focus on finding plans that avoid any need for general knowledge base reasoning at query execution time when sufficient cached results of earlier requests exist.

## 1 Introduction

Many types of structured data repositories can be viewed as collections of facts about objects (e.g., [Halevy *et al.*, 2006]), often with schematic information encoding rules and constraints for how objects relate to each other. An important class of search queries over these repositories are *object queries*, which return a set of object identifiers based on given selection criteria. The importance of these queries has given rise to a variety of formalisms for expressing object queries [Fletcher *et al.*, 2009; Jagadish *et al.*, 2007; Ross and Janevski, 2004].

In this paper, we consider a *description logic* (DL) based representation of the object retrieval problem. Description logics give us a formal representation of facts about objects and provide a rich framework for automated reasoning over the schema information accompanying object databases. In particular, we assume a database is given in the form of a *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ over some choice of DL dialect $\mathcal{L}$, in which $\mathcal{T}$ is a terminology (or TBox) that captures general knowledge, and in which $\mathcal{A}$ is a set of assertions (or ABox) that identifies objects of interest to some agent and asserts facts about those objects. In this setting, an object corresponds to an individual name occurring in an ABox and the problem of object querying corresponds to the well-known problem of *instance retrieval* [Horrocks *et al.*, 2004].

A particular instance retrieval problem for a knowledge base $\mathcal{K}$ over $\mathcal{L}$ is defined by a concept $C$ in $\mathcal{L}$. The problem is to compute all objects (or instances) $a$ occurring in $\mathcal{A}$ for which $\mathcal{K} \models a : C$, that is, for which the assertion $a : C$, stating that $a$ belongs to concept $C$, is a logical consequence of $\mathcal{K}$. Thus, object queries correspond to concepts in $\mathcal{L}$ and return the subset of ABox objects that satisfy this condition.

We consider a more general problem we call *assertion retrieval* in which a concept $C_a$ in $\mathcal{L}$ is now paired with each object $a$ returned by a query concept. $C_a$ will be "the most informative" concept about $a$ that can be expressed in a subset of $\mathcal{L}$ defined by an additional query parameter. In particular, we assume a user query is now given by a pair $(C, Pd)$ in which $C$ is a query concept in $\mathcal{L}$ serving the same role as in instance retrieval, and in which $Pd$ is the additional parameter, called a *projection description*, that defines a subset $\mathcal{L}_{Pd}$ of the concepts of $\mathcal{L}$. The query returns assertions of the form $a : C_a$ such that $\mathcal{K} \models a : (C \sqcap C_a)$, where $C_a$ is the *most specific concept* in $\mathcal{L}_{Pd}$ for which this condition holds. A projection description generalizes the effect of the relational *projection operation* by providing a more general way of controlling both the format and information content of query results.

There are two compelling reasons for considering assertion retrieval over basic object retrieval. The first is a practical reason that relates to usability: since it becomes possible to include relevant facts about objects in addition to their identifiers, the results of querying can be more informative and relevant to a user searching a knowledge base. The second is a more technical reason that relates to performance: by caching the computation of earlier assertion retrieval queries, it becomes possible to explore view-based query rewriting in the context of object queries over knowledge bases. In particular, we show rewritings that *eliminate the need for general knowledge base reasoning* during query processing by reusing previously computed query results.

A running example used in this work relates to a hypothetical knowledge base system encoding product data. In response to some user action, a web browser submits queries over the knowledge base. The queries originate in the web pages for an enterprise, and are replaced on the fly with the resulting set of concept assertions computed by the query (suitably mapped to HTML).

**Example 1** Consider the case of an online dealer of photography equipment. As part of a web presence, the dealer maintains (1) a knowledge base $\mathcal{K}$ with a TBox and ABox that respectively capture ontological knowledge of digital cameras and facts about particular cameras available for purchase through the dealer, and (2) a collection of web pages with embedded user queries over this knowledge base. For example, one of the web pages might have a query $Q$ with a query

concept $C$ of the form

$$ProductCode = \text{“digicam”} \sqcap Price < \text{“250.00”}$$

paired with a projection description $Pd$ of the form

$$(Name? \sqcap \exists Supplier.(OnlineAddr? \sqcap Rating?)).$$

Consequently, when browsing this page, a user sees in place of $Q$ a list of inexpensive digital cameras, with each list element displaying the name of the camera together with a nested sublist of supplier URLs and ratings for that camera.

The example above illustrates how a set of concept assertions computed by our query language can resemble a nested relation. Note that such queries are not expressible as conjunctive queries over either a DL-based knowledge base or a relational database. Our contributions are as follows.

1. We propose the task of *assertion retrieval* for object queries over knowledge in which it becomes possible for a user to control the format and content of additional facts about qualifying objects returned in response.
2. By choosing a DL-based presentation of object data in the form of a knowledge base, we provide a way to resolve the problem of incorporating ontological domain knowledge for object queries.
3. We introduce an assertion retrieval algebra over a combination of a knowledge base and a set of cached query results. We show how this can enable a transfer of basic database paradigms involving caching and view based query rewriting.
4. We present an optimization framework for query rewriting and show that applying rewrites to query plans can enable the resulting plans to avoid general knowledge base reasoning during their execution.

We also present experimental results illustrating evidence for both the utility of our optimization framework and the feasibility of our approach for supporting assertion retrieval over knowledge bases.

Subsequent sections are organized as follows. Section 2 provides the necessary foundations. Our main results follow in Section 3 in which we introduce our assertion retrieval algebra together with an optimization framework for this algebra that enables a transfer of basic database paradigms involving caching and view-based query rewriting. The section concludes by considering how rewrites can be introduced that will overcome the overhead of general knowledge based reasoning during query execution. In Section 4, we present the results of preliminary experiments and conclude in Section 5 with a discussion of how our work leads to a variety of possible extensions and future work.

### 1.1 On Expressiveness and Related Work

The notion of an object query language for object data was proposed in [Ross and Janevski, 2004] and further considered in [Fletcher *et al.*, 2009] for object data that includes links between objects. The latter work also exhibits a correspondence between an object query language proposed by the authors and a restricted semi-join algebra over two-column relational data. Thus, in this earlier work, object query languages are viewed as strictly less powerful than first order query languages such as the relational algebra.

## 2 Definitions

We assume web data is formally understood as a knowledge base over some choice of DL dialect $\mathcal{L}$. For the remainder of the paper, $\mathcal{L}$ will correspond to the dialect $\mathcal{ALC}(\mathbb{S})$ defined below. However, our results apply to any dialect that has $\mathcal{ALC}(\mathbb{S})$ as a fragment, such as the dialect that underlies OWL DL [W3C, 2009]. (This requirement can be relaxed slightly without harm by removing the need to support concept negation.)

**Definition 1 (Description Logic $\mathcal{ALC}(\mathbb{S})$)** *Let $\{A, A_1, \ldots\}$, $\{R, R_1, \ldots\}$, $\{f, g, f_1, \ldots\}$ and $\{a, b, \ldots\}$ denote countably infinite and disjoint sets of* concept names*,* role names*,* concrete features *and* individual names*, respectively. A* concept *is defined by:*

$$\begin{array}{rcll} C, D & ::= & \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid \exists R.C & \\ & \mid & f = k & (\textit{equality over } \mathbb{S}) \\ & \mid & f < g & (\textit{linear order over } \mathbb{S}) \end{array}$$

*where $k$ is a finite string. A constraint $\mathcal{C}$ is an* inclusion dependency *$C \sqsubseteq D$, a* concept assertion *$a : C$, or* role assertion *$R(a, b)$. A* knowledge base *$\mathcal{K}$ is a finite set of constraints, further divided into a TBox $\mathcal{T}$ of all the inclusion dependencies and an ABox $\mathcal{A}$ of concept/role assertions.*

*An interpretation $\mathcal{I}$ is a 2-tuple $(\triangle \uplus \mathbb{S}, \cdot^{\mathcal{I}})$ in which $\triangle$ is an* abstract domain *of objects and in which $\mathbb{S}$ is a disjoint* concrete domain *of finite strings. The interpretation function $\cdot^{\mathcal{I}}$ maps each concrete feature $f$ to a total function $(f)^{\mathcal{I}} : \triangle \to \mathbb{S}$, each role name $R$ to a relation $(R)^{\mathcal{I}} \subseteq (\triangle \times \triangle)$, each concept name $A$ to a set $(A)^{\mathcal{I}} \subseteq \triangle$, the "$=$" symbol to the equality relation on $\mathbb{S}$, the "$<$" symbol to the binary relation for an alphabetical ordering of $\mathbb{S}$, and a finite string $k$ to itself. The interpretation function is extended to all concepts in the standard way, e.g., $(f = k)^{\mathcal{I}} = \{o \in \triangle \mid (f)^{\mathcal{I}}(o) = k\}$, $(\neg C)^{\mathcal{I}} = \triangle - (C)^{\mathcal{I}}$, etc. An interpretation $\mathcal{I}$ satisfies an inclusion dependency $C \sqsubseteq D$ (concept assertion $a : C$, role assertion $R(a, b)$) if $(C)^{\mathcal{I}} \subseteq (D)^{\mathcal{I}}$ ($(a)^{\mathcal{I}} \in (C)^{\mathcal{I}}$, $((a)^{\mathcal{I}}, (b)^{\mathcal{I}}) \in (R)^{\mathcal{I}}$, respectively). We write $\mathcal{K} \models \mathcal{C}$ if all interpretations that satisfy constraints in $\mathcal{K}$ also satisfy $\mathcal{C}$.*

We use standard abbreviations such as $C \equiv D$ for $(C \sqsubseteq D) \sqcap (D \sqsubseteq C)$, $C \sqcup D$ for $\neg(\neg C \sqcap \neg D)$ and $f \leq k$ for $(f = k) \sqcup ((f < g) \sqcap (g = k))$. Also, given a finite set $S$ of $\mathcal{ALC}(\mathbb{S})$ concepts, we write $\sqcap S$ to be the concept $D_1 \sqcap \cdots \sqcap D_n$ for $D_i \in S$ (and $\top$ if $S = \emptyset$).

Now recall that a user query $(C, Pd)$ consists of a query concept $C$ paired with a projection description $Pd$. The syntax for a $Pd$ and the sublanguage of concepts in $\mathcal{ALC}(\mathbb{S})$ that are induced by a $Pd$ are defined as follows.

**Definition 2 (Projection Description)** *Let $f$, $R$ and $C$ be a concrete feature, role and concept, respectively. A* projection description *$Pd$ is defined by the grammar:*

$$Pd \ ::= \ C? \mid f? \mid Pd_1 \sqcap Pd_2 \mid \exists R.Pd \qquad (1)$$

**Definition 3 (Induced Concepts)** *Let $Pd$ be a projection description. We define the sets $\mathcal{L}_{Pd}$ and $\mathcal{L}^{\text{TUP}}_{Pd}$, the $\mathcal{L}$* concepts generated by *$Pd$ and $\mathcal{L}$* tuple concepts generated by

*Pd, respectively, as follows:*

$$\mathcal{L}_{Pd} = \{\sqcap S \mid S \subseteq_{\text{fin}} \mathcal{L}^{\text{TUP}}_{Pd}\}$$

$$\mathcal{L}^{\text{TUP}}_{C?} = \{C, \top\}$$
$$\mathcal{L}^{\text{TUP}}_{f?} = \{f = k \mid k \in \mathbb{S}\} \cup \{\top\}$$
$$\mathcal{L}^{\text{TUP}}_{Pd_1 \sqcap Pd_2} = \{C_1 \sqcap C_2 \mid C_1 \in \mathcal{L}^{\text{TUP}}_{Pd_1}, C_2 \in \mathcal{L}^{\text{TUP}}_{Pd_2}\}$$
$$\mathcal{L}^{\text{TUP}}_{\exists R.Pd_1} = \{\exists R.C \mid C \in \mathcal{L}_{Pd_1}\}$$

Thus, for a given $Pd$, any concept occurring in $\mathcal{L}_{Pd}$ satisfies a syntactic format *conforming* to $Pd$ independently of any knowledge base $\mathcal{K}$. For example, let $Pd = (A? \sqcap (f = 1)?)$. Then $\mathcal{L}_{Pd}$ is the set of concepts

$$\{\sqcap S \mid S \subseteq_{\text{fin}} \{(\top \sqcap \top), (A \sqcap \top), (\top \sqcap (f = 1)), (A \sqcap (f = 1))\}\}$$

where $\subseteq_{\text{fin}}$ denotes a finite subset. Given a knowledge base $\mathcal{K}$ and set of concepts $S$, we will need to refer to the *most informative* concepts in $S$ with respect to $\mathcal{K}$.

**Definition 4 (Most Informative Concepts w.r.t. $\mathcal{L}_{Pd}$)** *Let $S$ be a set of concepts specified by $\mathcal{L}_{Pd}$ and $\mathcal{K}$ knowledge base. We write $\lfloor S \rfloor_{\mathcal{K}}$ to denote $\{C \in S \mid \neg\exists D \in S : (\mathcal{K} \models D \sqsubseteq C, \mathcal{K} \not\models C \sqsubseteq D)\}$.*

**Lemma 5** *Let $\mathcal{K}$ be an $\mathcal{ALC}(\mathbb{S})$ knowledge base, $Pd$ a projection description and $C$ a concept. The following hold for the set $S$ of concepts defined by $\{D \in \mathcal{L}_{Pd} \mid \mathcal{K} \models C \sqsubseteq D\}$:*

1. *$\lfloor S \rfloor_{\mathcal{K}}$ is non-empty;*
2. *$\mathcal{K} \models C_1 \equiv C_2$, for any $\{C_1, C_2\} \subseteq \lfloor S \rfloor_{\mathcal{K}}$; and*
3. *$\lfloor \lfloor S \rfloor_{\mathcal{K}} \rfloor_{\emptyset}$ is non-empty,*

Parts *1* and *2* of Lemma 5 ensure that at least one least subsuming concept of $C$ exists in $\mathcal{L}_{Pd}$ and, when there is more than one, that any pair are semantically equivalent with respect to a given $\mathcal{K}$. Note that some such $\mathcal{L}$ restriction of $\mathcal{ALC}(\mathbb{S})$ is essential to ensure Part *1*, e.g., that a more general fragment that simply excludes concept negation from $\mathcal{ALC}(\mathbb{S})$ may not have this property [Baader *et al.*, 2007]. Also note that, although $\mathcal{L}_{Pd}$ is infinite in general, for any fixed and finite terminology $\mathcal{T}$ and concept $C$, the language $\mathcal{L}_{Pd}$ restricted to the symbols used in $\mathcal{T}$ and $C$ is necessarily finite. Part *3* of Lemma 5 ensures that, among the least subsuming concepts of $C$ in $\mathcal{L}_{Pd}$ with respect to $\mathcal{K}$, there is at least one least subsuming concept that is *the most informative* when no knowledge of $\mathcal{K}$ is presumed. For example, let $\mathcal{K} = \{A \sqsubseteq (f = 1)\}$ and $Pd = (A? \sqcap f?)$, and let $S = \{D \in \mathcal{L}_{Pd} \mid \mathcal{K} \models A \sqsubseteq D\}$. Then

1. $\lfloor S \rfloor_{\mathcal{K}} = \{A \sqcap (f = 1), A \sqcap \top\}$,
2. $\lfloor \lfloor S \rfloor_{\mathcal{K}} \rfloor_{\emptyset} = \{A \sqcap (f = 1)\}$ and
3. $\lVert S \rVert_{\mathcal{K}} = A \sqcap (f = 1)$.

where $\lVert S \rVert_{\mathcal{K}}$ is the minimum concept from a set of concepts $\lfloor \lfloor S \rfloor_{\mathcal{K}} \rfloor_{\emptyset}$ according to an (arbitrary) total ordering of all concepts in $\mathcal{ALC}(\mathbb{S})$.

**Definition 6 (Query Semantics)** *Let $\mathcal{K}$ be an $\mathcal{ALC}(\mathbb{S})$ knowledge base and $Q = (C, Pd)$ a user query over $\mathcal{K}$. Then $Q$ computes the set of concept assertions*

$$\{a : \lVert \{D \mid D \in \mathcal{L}_{Pd}, \mathcal{K} \models a : D\} \rVert_{\mathcal{K}} \mid \mathcal{K} \models a : C, a \text{ occurs in } \mathcal{K}\} \quad (2)$$

Observe that concept assertion retrieval generalizes instance retrieval. In particular, an instance retrieval query $C$ over $\mathcal{K}$ can be formulated as query $(C, \top?)$ (effectively retrieving no further information about qualifying individual names in $\mathcal{K}$).

# 3 An Assertion Retrieval Algebra

We now introduce an *algebra* for manipulating sets of concept assertions. The algebra is centered around our ability to store (cache) and later use results of previous queries to aid the evaluation of subsequent queries. To this end the algebra uses operations for (efficient) selection of qualifying objects from cached query results [Pound *et al.*, 2007] and for concept projection [Pound *et al.*, 2009]. Additional operators are included that allow basic combinations of queries. We show how expressions in this algebra can describe a variety of *query plans* for evaluating a user query that can vary widely in the cost of their evaluation, and we outline how several standard relational-style *query optimization techniques* can be accommodated in this framework.

## 3.1 Cached Query Results

Intuitively, *cached query results* store explicit information about the qualifying individuals obtained by previous user queries.

**Definition 7 (Cached Query Results)** *A* cached query result *$S_i$ is the set of concept assertions obtained by a user query $(C_i, Pd_i)$.*

The above definition permits the existence of any number of cached query results often organized in data structures designed to support user queries[1]. Note that cached query results are *essential* in our approach. They enable query evaluation to avoid (or reduce) the amount of general DL reasoning during query evaluation. A particular query answer may be more preferable for caching than that of another, though, in principle, all user query results could be cached. Details on selecting which query answer to store are beyond the scope of this paper.

**Example 2** To continue with our running example: we assume cached query results are available:

$S_1$ := $(\top, ProductCode?)$,
$S_2$ := $(Price <$ "999.99", $Price?)$, and
$S_3$ := $(\top, Name? \sqcap \exists Supplier.(OnlineAddr? \sqcap Rating?))$.

The first caches the $ProductCode$ of every product (with the idea that the underlying data structure can be efficiently searched given a particular product code), the second caches the $Price$ for products costing under $1000, and the last stores a more elaborate *projection* of the information associated with every product.

## 3.2 The Algebra

To facilitate efficient evaluation of such requests we introduce a query algebra to manipulate sets of concept assertions. The algebra allows for the use of cached query results to speed-up search for qualifying individuals and to retrieve appropriate concepts needed to construct answer concept assertions.

[1] This is analogous to relational systems, multiple specialized indices are typically defined to support queries. This is in contrast to approaches that aspire to developing a "universal" search structure to represent semantic data.

**Definition 8 (Assertion Retrieval Algebra)** *The algebra consists of the operators listed in the grammar below.*

$$
\begin{array}{lll}
Q ::= C & \{a:C \mid a \text{ appears in } \mathcal{K}\} \\
\mid\ P^{\mathcal{K}} & \{a:\top \mid a \text{ appears in } \mathcal{K}\} \\
\mid\ S_i(Q) & \{a:C \mid (a:C)\in S_i, (a:D)\in Q, \\
 & \qquad \{a:C\} \models a:D\} \\
\mid\ \sigma^{\mathcal{K}}_C(Q) & \{a:D \mid (a:D)\in Q, \mathcal{K}\cup\{a:D\}\models a:C\} \\
\mid\ \pi^{\mathcal{K}}_{Pd}(Q) & \{a:[\![\{D \mid \mathcal{K}\cup\{a:C\}\models a:D, \\
 & \qquad D\in\mathcal{L}_{Pd}\}]\!]_{\mathcal{K}} \mid (a:C)\in Q\} \\
\mid\ Q_1\sqcap Q_2 & \{a:D_1\sqcap D_2 \mid (a:D_i)\in Q_i, i=1,2\}
\end{array}
$$

*where $C$ is a concept description and $S_i$ a cached query result. The semantics of each operator is given immediately to the right of its position in the grammar, and is defined with respect to (a) zero or more cached query results $\{S_1,\ldots,S_n\}$, and (b) a given knowledge base $\mathcal{K}$.*

*The operators are either* pure *or* impure*: $C$ is impure, $P^{\mathcal{K}}$ and $S_i(Q)$ are pure, $\sigma^{\mathcal{K}}_C(Q)$ and $\pi^{\mathcal{K}}_{Pd}(Q)$ are pure if $Q$ is pure, and $Q_1\sqcap Q_2$ is pure if both $Q_1$ and $Q_2$ are pure.*

Each of the operators maps sets of concept assertions to a set of concept assertions, with the $S_i(Q)$ operator providing an access to the previously cached results of queries. The operator accepts an argument that can supply *search conditions* with the intention of using the underlying data structure representing the cached result to facilitate efficient search. For example, the cached result $S_1$ from Example 2 can be used to efficiently search for descriptions of the form $(ProductCode = k)$ for a given string $k$.

Note that queries of the form $C$ can be, without loss of generality, *compactly represented* by a single generalized assertion of the form $\{\star : C\}$ where $\star$ stands for an arbitrary ABox individual. This arrangement leads to greatly improved efficiency when executing $S_i(C)$ operators to select cached results subsumed by the constant description $C$.

### 3.3 Queries as Algebraic Expressions

In this setting, a given user query $(C, Pd)$ can always be translated into the algebraic expression

$$\pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_C(P^{\mathcal{K}})); \tag{3}$$

this follows immediately from the definitions. However, to benefit from the performance gains made possible by cached query results, the algebra allows a richer space of expressions:

**Lemma 9 (Caching Introduction)** *Let $(C, Pd)$ be a user query. Then the expression*

$$\pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_C((S_1(C_1)\sqcap\cdots\sqcap S_n(C_n)))) \tag{4}$$

*is equivalent to (3), provided that (i) $S_i := (D_i, Pd_i)$), (ii) $\mathcal{K}\models C\sqsubseteq(D_1\sqcap ...\sqcap D_n)$, and (iii) $C_i = [\![\{D \mid D\in\mathcal{L}_{Pd_i}, \mathcal{K}\models C\sqsubseteq D\}]\!]_{\mathcal{K}}$, for all $0<i\leq n$.*

Conditions (i) and (ii) ensure that the combination (intersection) of the cached query results $S_i$ contains sufficient data to answer the original query. The last condition in the lemma serves two purposes: first it supplies a sufficiently *general* search concept to each of the cached results. Note that using the original search concept $C$ instead would lead to loss of answers since the concept assertions stored in the cached results are more general than those implied by the knowledge base, in general. Secondly, as the concepts $C_i$ are results of the *same projections* used to create the cached result $S_i$, all the general subsumption checks reduce to *structural tests* (cf. section 3.4). This rewriting *completely avoids* the use of $P^{\mathcal{K}}$.

The rewriting, when coupled with the ability to store and search efficiently among descriptions, yields a path to defining appropriate physical data layout designs in the form of *cached query results* and in turn to *efficient plans for answering assertion retrieval queries*. The general form of (4) can be further simplified using the analogues of relational-style query rewrites that allow the use of cached query results:

**Lemma 10 (Removing Redundant Selections)** *The selection operation $\sigma^{\mathcal{K}}_C(\cdot)$ can be removed from (4) to obtain the expression*

$$\pi^{\mathcal{K}}_{Pd}((S_1(C_1)\sqcap\cdots\sqcap S_n(C_n))) \tag{5}$$

*if $\mathcal{K}\models(C_1\sqcap\ldots\sqcap C_n)\sqsubseteq C$.*

**Example 3** Recall the running example query (1). With the help of cached query results defined in Example 2, we can obtain the following equivalent query expression:

$$\begin{array}{l}\pi^{\mathcal{K}}_{Pd}(S_1(ProductCode = \text{“digicam”})\sqcap \\ \qquad S_2(Price < \text{“250.00”})\sqcap S_3(\top)).\end{array} \tag{6}$$

The cached results $S_1$ and $S_2$ fully qualify the individuals needed to answer the query and can be efficiently accessed using the concepts $(ProductCode = \text{“digicam”})$ and $(Price < \text{“250.00”})$, respectively. The reason for using $S_3$ is to form the concept assertions for the answer since $S_3$ stores the most specific descriptions conforming to $Pd$.

A final feature of the proposed algebraic framework is its ability to use *results of queries* to further qualify *scans over cached query results*:

**Lemma 11 (Nested Searches)** *An expression of the form*

$$\pi^{\mathcal{K}}_{Pd'}(S_i(C_i)\sqcap S_j(C_j))$$

*is equivalent to*

$$\pi^{\mathcal{K}}_{Pd'}(S_i(C_i\sqcap\pi^{\mathcal{K}}_{Pd}(S_j(C_j))))$$

*where $Pd$ is such that $\mathcal{L}_{Pd}\subseteq\mathcal{L}_{Pd_i}$ (the projection description used to define $S_i$) and $Pd'$ is an arbitrary projection.*

Note that the projection description $Pd$ in the above rewriting rule is not unique; in particular, the projection description $\top?$ can be always used in place of $Pd$. However, the more general this projection description is, the less information about objects retrieved from $S_j$ is used to qualify the search in $S_i$.

**Example 4** The above lemma applied twice to our running example (6) will produce the following expression:

$$\begin{array}{l}\pi^{\mathcal{K}}_{Pd}(S_3(\pi^{\mathcal{K}}_{\top?}(S_2(Price < \text{“250.00”}\sqcap \\ \qquad \pi^{\mathcal{K}}_{\top?}(S_1(ProductCode = \text{“}digicam\text{”})))))))\end{array} \tag{7}$$

Note that the outermost projection is identical to the projection used to define $S_3$ and can be removed.

### 3.4 On Purely Structural Reasoning

There are a variety of cases in which the operators in our algebra can be evaluated with simple structural subsumption testing in place of general reasoning (indeed, this is already the case when accessing the cached results). In this section, we characterize a general condition in which this holds for various operators in a given concept assertion query $Q$. We devise a test that allows us to determine whether a particular operator of our algebra can be *executed* without referring to the knowledge base $\mathcal{K}$. This goal requires us to devise conditions under which $\mathsf{OP}^{\mathcal{K}}(Q_1,\ldots,Q_k) = \mathsf{OP}^{\emptyset}(Q_1,\ldots,Q_k)$

for every operator OP in our algebra and every knowledge base $\mathcal{K}$, where $\mathsf{OP}^{\emptyset}$ denotes executing the operator w.r.t. the empty knowledge base. Intuitively this means that the concept assertions in the answers to $Q_i$ contain sufficient *explicit* information about an individual; the goal is to ultimately obtain this information using some cached query result rather than via reasoning in $\mathcal{K}$.

**Definition 12 (Representative Language)** *We define $\mathcal{L}_Q$ to be a language of representative concepts for an algebraic query $Q$ as follows:*

$$\mathcal{L}_Q = \begin{cases} \{C\} & \text{if } Q = \text{“}C\text{”}; \\ \{\top\} & \text{if } Q = \text{“}P^{\mathcal{K}}\text{”}; \\ \mathcal{L}_{Pd_i} & \text{if } Q = \text{“}S_i(Q_1)\text{”}; \\ \mathcal{L}_{C \sqcap Q_1} & \text{if } Q = \text{“}\sigma^{\mathcal{K}}_C(Q_1)\text{”}; \\ \mathcal{L}_{Pd} & \text{if } Q = \text{“}\pi^{\mathcal{K}}_{Pd}(Q_1)\text{”}; \text{ and} \\ \{C \sqcap D \mid C \in \mathcal{L}_{Q_1}, D \in \mathcal{L}_{Q_2}\} & \text{if } Q = \text{“}Q_1 \cap Q_2\text{”}, \end{cases}$$

*where $Pd_i$ is the projection description used to define $S_i$, For representative languages $\mathcal{L}_{Q_1}$ and $\mathcal{L}_{Q_2}$ we define $\mathcal{L}_{Q_1} \hookrightarrow \mathcal{L}_{Q_2}$ if for all $C_1 \in \mathcal{L}_{Q_1}$ there is $C_2 \in \mathcal{L}_{Q_2}$ such that $\models C_1 \equiv C_2$.*

**Theorem 13** *Let $(C, Pd)$ be a user query, $(C_i, Pd_i)$ queries that define cached query results $S_i$, and $Q$ an algebraic query equivalent to $(C, Pd)$. Then, for every pure subquery $Q'$ of $Q$, the following holds:*

1. *$\sigma^{\mathcal{K}}_C(Q') = \sigma^{\emptyset}_C(Q')$ iff $\mathcal{L}_{\sigma^{\mathcal{K}}_C(Q')} \hookrightarrow \mathcal{L}_{Q'}$;*
2. *$\pi^{\mathcal{K}}_{Pd}(Q') = \pi^{\emptyset}_{Pd}(Q')$ iff $\mathcal{L}_{\pi^{\mathcal{K}}_{Pd}(Q')} \hookrightarrow \mathcal{L}_{Q'}$.*

*Proof (sketch): (a sketch for case 1; the other case is similar)* For $(a : D) \in \sigma^{\mathcal{K}}_C(Q')$ we have $(a : D) \in Q'$ and $\mathcal{K} \cup \{a : D\} \models a : C$ where $D$ is the most specific in $\mathcal{L}_{Q'}$ such that $a : D$. However, by the assumption of the theorem, concept $C \sqcap D$, up to equivalence, is also a member of $\mathcal{L}_{Q'}$. Thus, $\not\models D \sqsubseteq C$ implies $a : C \sqcap D \in Q'$: a contradiction. Hence $C \sqsubseteq D$ and thus $(a : C) \in \sigma^{\emptyset}_C(Q')$. The other direction is immediate from definitions.

To make the above theorem into an effective rewriting rule we need to define finite approximations of the languages $\mathcal{L}_Q$.

**Definition 14** *Let $(C, Pd)$ be a user query and $(C_i, Pd_i)$ queries that define cached query results $S_i$. We define a set of concrete domain values (strings) $S$ to be the set of values $k$ such that $(f = k)$ is a subexpression in any of the above queries for some feature $f$. We define $\mathcal{L}^{\text{fin}}_Q$ to be the restriction of the language $\mathcal{L}_Q$ to those concepts $C$ for which $k \in \{\star\} \cup S$ whenever $f = k$ is a subconcept of $C$, where $\star$ is an arbitrary concrete value that does not appear in $S$.*

It is easy to see that $\mathcal{L}^{\text{fin}}_Q$ is finite; moreover:

**Lemma 15** *$\mathcal{L}_{Q_1} \hookrightarrow \mathcal{L}_{Q_2}$ iff $\mathcal{L}^{\text{fin}}_{Q_1} \hookrightarrow \mathcal{L}^{\text{fin}}_{Q_2}$.*

Hence the precondition for the rewriting defined in Theorem 13 can be tested using finitely many subsumption tests in the underlying description logic.

**Example 5** Applying the results in Theorem 13 to the algebraic version of our running example (7) top-down yields:

$$S_3(\pi^{\emptyset}_{\top?}(S_2(Price < \text{“}250.00\text{”} \cap \pi^{\emptyset}_{\top?}(S_1(ProductCode = \text{“}digicam\text{”}))))) \quad (8)$$

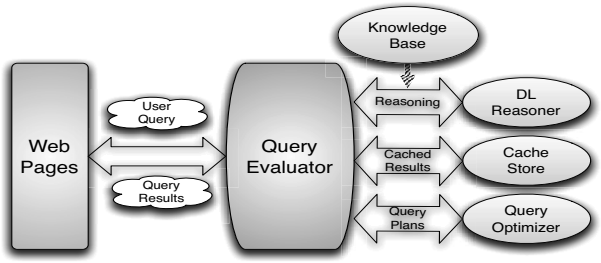


Figure 1: System Architecture

The complete query can now be executed exclusively using the cached query results and without referring to the knowledge base $\mathcal{K}$.

Note the essential use of $S_3$ that stores sufficient explicit information about retrieved objects, even though in itself does not facilitate the search for such objects.

## 4 Experimental Evaluation

To evaluate the utility of our optimization framework and the feasibility of our approach to supporting object queries over knowledge bases, we implemented the assertion retrieval algebra defined in the previous section together with a basic DL reasoner to support subsumption testing in $\mathcal{ALC}(\mathbb{S})$. Our implementation uses standard tableaux-based techniques with a straightforward extension to support reasoning over our very simple concrete domain of strings. Our implementation of the projection and scanning operations on top of the DL reasoner are based on the algorithms reported in [Pound *et al.*, 2007] and [Pound *et al.*, 2009].

An overview of our system architecture is illustrated in Figure 1. The system assumes clients browse Web pages that contain embedded user queries. These are more generic queries that have *parameters* that are initialized by user input when browsing such pages. The resulting queries are then sent to the query evaluator for execution. In the current system, general plans for the original queries with parameters will have already been cached. These plans use cached query results that are stored in specialized data structures called *description indices* [Pound *et al.*, 2007]. Initialized queries are then evaluated using the cached plans to produce the resulting set of concept assertions, which are then returned to the client browser for display.

We compare two systems, both of which evaluate queries using cached query results. The baseline system always evaluates the query using calls to our DL reasoner. The optimized system takes advantage of the structure of cached queries to use structural subsumption testing and avoid general DL reasoning in some cases. The full source code for our implementation, along with our evaluation workload and test data is available online[2]. For our evaluation knowledge base, we encoded all publications from the last ten years of all program committee members for major conferences as an ABox (approximately 900 publications). We then extracted a taxonomy of computer science conferences from Wikipedia using an automated procedure, and hand coded additional axioms for affiliations of authors for our TBox (approximately 350 inclusion dependencies about authors' affiliation, organization locations and so on). Finally, we constructed a suite

[2]http://code.google.com/p/projection-alcd/

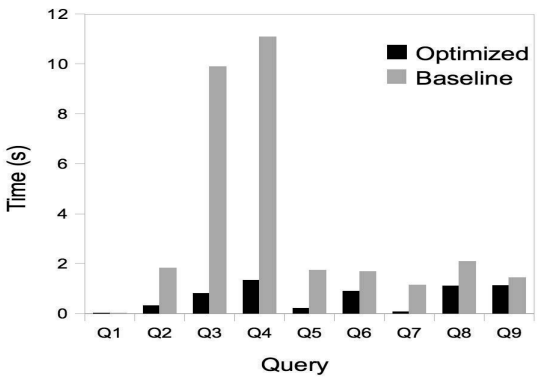

Figure 2: Execution time of queries.

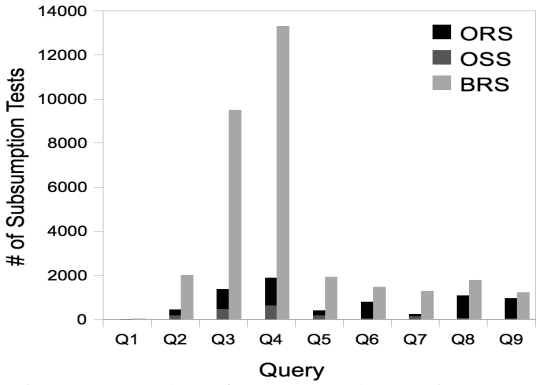

Figure 3: Number of reasoner subsumption tests.

of queries that exercise various scenarios, such as feature equality queries, feature range queries, and arbitrary concept queries including disjunctions and existential quantification. Queries were created for scenarios in which a cached query result efficiently supports the query (no general DL reasoning is required) and for scenarios in which a full scan and general DL reasoning is necessary. For example, one of our queries finds all publications published in some artificial intelligence proceedings, and returns concept assertions containing the title, proceedings name, and author names:

$$(\exists inProceedings.Artificial\_intelligence,\ title? \sqcap (\exists inProceedings.name?) \sqcap (\exists hasAuthor.name?)).$$

Figures 2 and 3 summarize our query processing experiments. The results are evidence for both the utility of our optimization framework and the feasibility of our approach to supporting object queries over knowledge bases. Figure 2 shows the overall execution time of each system over all queries with and without optimization, and Figure 3 shows a breakdown of the number of structural (OSS) and reasoner (ORS) subsumption tests for the optimized system, compared to the total number of reasoner subsumption tests for the baseline system (BRS). The results indicate that the optimized system has a clear performance advantage across a majority of our test queries. In particular, queries 3 and 4 involve feature projections on multiple features for a large number of results. In these scenarios, $\mathcal{K}$-free structural subsumption testing leads to a significant advantage over the baseline.

## 5 Conclusions & Future Work

The framework for assertion retrieval proposed in this paper provides a basis for introducing efficient relational-style query processing to querying object data. The key features of the approach are the ability to compute *projections* of general concepts to make properties of individuals syntactically explicit, to cache such assertions for efficient query evaluation, and to account for circumstances in which general knowledge base reasoning can be supplanted with simple structural subsumption testing at query evaluation time.

Future research can use the proposed query algebra to develop additional tools and techniques facilitating efficient query execution, for example: (1) optimization techniques that determine optimal reformulations of user queries in the algebra or its extensions, possibly integrated with cost and selectivity estimates; and (2) extensions of the language of projection descriptions that allow, e.g., alternatives in the explicit information stored with objects (efficient query processing in such an extension crucially depends on the ability to reason with fragments of the original knowledge base, possibly based on the notion of modules [Konev *et al.*, 2008]). Another direction of research is whether more complex user queries, e.g., an equivalent of conjunctive queries, can be accommodated by modest extensions to the framework.

## References

[Baader *et al.*, 2007] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. App. Logic*, 5(3):392–420, 2007.

[Fletcher *et al.*, 2009] George H. L. Fletcher, Jan Van den Bussche, Dirk Van Gucht, and Stijn Vansummeren. Towards a theory of search queries. In *Proc. ICDT*, pages 201–211, 2009.

[Halevy *et al.*, 2006] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *Proc. PODS*, pages 1–9, 2006.

[Horrocks *et al.*, 2004] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The Instance Store: DL Reasoning with Large Numbers of Individuals. In *Proc. Description Logics*, 2004.

[Jagadish *et al.*, 2007] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proc. ACM SIGMOD*, pages 13–24, 2007.

[Konev *et al.*, 2008] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. Semantic modularity and module extraction in description logics. In *Proc. ECAI 2008*, pages 55–59, 2008.

[Pound *et al.*, 2007] Jeffrey Pound, Lubomir Stanchev, David Toman, and Grant Weddell. On Ordering Descriptions in a Description Logic. In *Proc. Description Logics*, pages 123–134. CEUR-WS vol. 250, 2007.

[Pound *et al.*, 2009] Jeffrey Pound, David Toman, Grant Weddell, and Jiewen Wu. Concept Projection in Algebras for Computing Certain Answer Descriptions. In *Proc. Description Logics*. CEUR-WS vol. 477, 2009.

[Ross and Janevski, 2004] Kenneth A. Ross and Angel Janevski. Querying faceted databases. In *Proc. Semantic Web and Databases*, pages 199–218, 2004.

[W3C, 2009] W3C. OWL 2 Web Ontology Language: Document Overview, 2009. http://www.w3.org/TR/owl2-overview.