On Efficient Access to Knowledgebases

Lubomir Stanchev
Indiana University - Purdue University Fort Wayne

Abstract

With the growth of the Internet and the new vision for Web 3.0, ontology knowledgebases have become extremely popular. The hope is that one day the content and services on the Internet can be searched based on their semantics rather than based on keyword matching. This article shows first steps towards how a OWL-DL based knowledgebase can be accessed efficiently. We address challenges related to producing query results that contain inferred data and index structures that can support efficient execution.

1 Introduction

The Semantic Web has been a hot research topic for the last few years. John Markoff coined the phrase Web 3.0 in the New York Times in 2006 referring to providing semantic capabilities that enhance the user experience. Two main models: RDF (RDF) and OWL-DL (OWL) have been proposed to describe web resources, such as html and xml pages, web services, videos and images, to name a few. Query languages such as SPARQL and RQL have been proposed for querying RDF knowledgebases. Their implementation relies on representing RDF graphs as subject-predicate-object triplets in a relational database. The shortfall of this approach is that it does not support including inferred knowledge in the query result. Inference is supported in OWL-DL knowledgebases. However, research on efficient access to such knowledgebases has been limited.

Consider a user searching for a picture of a chair from the European Renaissance. The knowledgebase may contain the description of different chairs that were crafted during the period. However, without the knowledge that the European Renaissance took place between the 14th and 17th century and without the capabilities to reason with this knowledge, the query result will be empty. We therefore believe that the vision of Web 3.0 can be only realized by considering the available knowledgebases and domain ontologies in conjunction with reasoning capabilities.

This problem is challenging because queries over OWL-DL knowledgebases, unlike queries over RDF, cannot be easily translated to queries over a structured or semi-structured database schema, such as relational, object-oriented, or XML. Previous efforts have concentrated in two directions: querying RDF knowledgebases and reasoning with OWL-DL knowledgebases. However, little effort has been spent on supporting OWL-DL knowledgebase queries through efficient search structures. While papers like (Haarslev, Möller, and Wessel 2004) and (Sirin and Parsia 2007) propose languages for querying OWL-DL knowledgebases, they do not explain how to efficiently support the execution of such queries.

In this paper we build on the previous work in (Pound et al. 2008) and (Pound et al. 2007) that focuses on defining a query language for OWL-DL and building efficient data structures to support it.

In the next section we present an overview of existing approaches to searching the web and outline the advantages of our approach. Our key contributions in this paper are describing a new language for querying OWL-DL ontologies in Section 3 and describing index structures that can support efficient execution of OWL-DL queries in Section 4.

2 Searching in the Internet

There are more than one hundred million web pages on the Internet and searching in them is analogous to searching for a needle in a haystack. Most search engines use web crawlers to populate their database and then process text queries on the stored data (Lawrence and Giles 1999). Either the whole text of web pages or their title and selected paragraphs are indexed. Companies that create and support web search engines usually make profit by providing appropriate advertising. Again, index structures for text fields, such as Patricia Trees (Morrison 1968), are used to index the data. As a consequence, if a user searches for cars, then they will not get a link to the web page for Chrysler, unless the title or the indexed text of the web pages contains the word "car" or some derivation of it. Another problem is matching web pages with advertisements. For example, a user may be reading a web page on tree data structures and see advertisements for natural tree furniture based on keyword matching.

In our proposal, a web resource (such as a web page, web service, RSS feed, to name a few) will be described as a concept description. Of course, we do not expect a casual user to be knowledgable enough to specify complex concept descriptions against existing ontologies. We therefore propose that a Web Master can describe a web resource by answering a serious of guided questions to navigate through existing ontologies. Similarly, a user that is searching for a web resource will answer a sequence of guided questions to specify the content they are interested in. For example, if a user searches for a car, then they will be asked if they are interested in real or model cars, whether they are interested in foreign or domestically manufactured cars, and so on. Then the query that is constructed from the answers to the questions will be matched against the knowledgebase and all concepts that have descriptions that subsume the user request will be returned. Similarly, an advertiser can spend time answering a series of guided questions to describe the content of web pages they are interested in advertising on.

We believe that our approach will significantly improve the semantic quality of the search result. The disadvantage is that the queries are now much more complex and an inference engine, such as FaCT++ (Tsarkov and Horrocks 2006) or Racer (Haarslev and Möller 2003), needs to be invoked during the query execution.

3 The Query Language

In this subsection we describe our query language. Our knowledgebase consists of a set of objects, where a concept description is associated with every object. A query will have two parts: a concept description describing the query result and an ordering description describing the ordering of the result. All objects that have descriptions that satisfy (i.e., are subsumed by) the query concept description will be returned in the order specified by the ordering description.

3.1 Concept Description Language

For demonstration purposes, we will use the description logic dialect $\mathcal{ALCQ}(\mathcal{D})$ (Baader et al. 2003) to specify concept descriptions, where this choice is rich enough to illustrate our approach. However, the described approached can be applied to any other description logic dialect with a total linearly ordered domain.

We will use $\{C, D, \ldots\}$ to define a primitive concepts, $\{R, S, \ldots\}$ for roles, $\{f, g, \ldots\}$ for concrete features, k for constants, and n for integers.

Definition 1 (Description Logic $\mathcal{ALCQ}(\mathcal{D})$) *An arbitrary concept can then be defined by the following grammar:*

$D ::= f < g \mid f < k \mid f = k \mid C \mid D \sqcup E \mid \neg D \mid \exists R.D \mid (> n\ R\ D) \mid (= n\ R\ D)$.

A concept is defined relative to an interpretation $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \Delta_C, .^{\mathcal{I}} \rangle$, where $\Delta_{\mathcal{I}}$ is a an abstract domain, Δ_C is a linearly ordered concrete domain, and $.^{\mathcal{I}}$ is an interpretation function that maps each primitive concept to a subset of $\Delta_{\mathcal{I}}$, each constant k to an element of Δ_C, each concrete feature to a total function from $\Delta_{\mathcal{I}}$ to Δ_C, and each role to a binary relation: $\Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$. The interpretation function is extended to arbitrary concepts in the following way (Baader et al. 2003).

$(f < g)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, f^{\mathcal{I}}(x) < g^{\mathcal{I}}(x)\}$
$(f < k)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, f^{\mathcal{I}}(x) < k^{\mathcal{I}}\}$
$(f = k)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, f^{\mathcal{I}}(x) = k^{\mathcal{I}}\}$
$(D \sqcup E)^{\mathcal{I}} ::= D^{\mathcal{I}} \cup E^{\mathcal{I}}$
$(\neg D)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, x \notin D^{\mathcal{I}}\}$
$(\exists R.D)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, \exists y \in D^{\mathcal{I}} \text{ and } (x, y) \in R^{\mathcal{I}}\}$
$(> n\ R\ D)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, |\{y \in D^{\mathcal{I}} | (x, y) \in R^{\mathcal{I}}\}| > n\}$
$(= n\ R\ D)^{\mathcal{I}} ::= \{x \in \Delta_{\mathcal{I}}, |\{y \in D^{\mathcal{I}} | (x, y) \in R^{\mathcal{I}}\}| = n\}$

Note that we have used $|\cdot|$ to denote the number of items in the set. We will use the following shortcuts: $f = g$ to denote $\neg((f < g) \sqcup (g < f))$, $D \sqcap E$ to denote $\neg(\neg D \sqcup \neg E)$, $(\leq n\ R\ D)$ to denote $\neg(> n\ R\ D)$, $\perp$ to denote $D \sqcap \neg D$, $\top$ to denote $D \sqcup \neg D$, etc.

For example, a chair can have the description $(= 3\ has\_support\ leg) \sqcap (century\_crafted = 15 \sqcup century\_crafted = 16) \sqcap (\exists color.red) \sqcap (date\_last\_sold < date\_photo\_taken)$. Every object that has this description will be supported by three legs, will be crafted in the 15th or 16th century, at least one of its colors will be red, and a picture of the object was taken after the object was last sold.

A terminology $\mathcal{T}$ consists of a set of inclusion dependencies of the form $D \sqsubseteq E$ (read D is subsumed by E). We will say that that the inclusion dependency $D \sqsubseteq E$ is a consequence of the terminology $\mathcal{T}$ and write $\mathcal{T} \models D \sqsubseteq E$ if and only if for any interpretation $\mathcal{I}$ of $\mathcal{T}$, $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. An example of a subsumption that can be used to correctly answer the query in the introduction is $EuropeanRenaissance \sqsubseteq centry \geq 14 \sqcap century \leq 17$, which means that if something is described as belonging to the European Renaissance, then its value for the *century* concrete feature must be between 14 and 17.

3.2 Ordering Description Language

We will use D^* to denote the description D where all artifacts are renamed by attaching a * at the end of their names.

Definition 2 (ordering description) *An ordering description Od has the following syntax.*

$Od ::= \epsilon \mid f : Od \mid \{D_1 : Od_1, \ldots, D_m : Od_m\}$

In order for this ordering description to be valid relative to a terminology $\mathcal{T}$, it must be the case that $\mathcal{T} \models D_i \sqcap D_j \sqsubseteq \perp$ for $1 \leq i \neq j \leq m$ and $\mathcal{T} \models \top \sqsubseteq \bigsqcup_{i=1}^{m} D_i$. We will say that the concept description D is before E relative to an ordering description Od and terminology $\mathcal{T}$ and write $D \prec_{Od,\mathcal{T}} E$ in the following cases:

- *when $Od = f : Od_1$:*
  $\mathcal{T} \cup \mathcal{T}^* \models (D \sqcap E^*) \sqsubseteq (f < f^*)$ *or*
  $\mathcal{T} \cup \mathcal{T}^* \models (D \sqcap E^*) \sqsubseteq (f = f^*)$ *and* $D \prec_{Od_1,\mathcal{T}} E$
- *when $Od = \{D_1 : Od_1, \ldots, D_m : Od_m\}$:*
  $\mathcal{T} \cup \mathcal{T}^* \models (D \sqsubseteq D_i)$ *and* $\mathcal{T} \cup \mathcal{T}^* \models (E \sqsubseteq D_j)$ *and $i < j$ for some i and j or*
  $\mathcal{T} \models (D \sqcap E) \sqsubseteq D_i$ *for some i and* $D \prec_{Od_i,\mathcal{T}} E$

Note that ϵ represents an empty ordering and will be omitted when clear from the context. The ordering description language is very rich. To demonstrate its capabilities, consider a user who is looking to buy a fuel efficient car, but also cares about the manufacturer of the car and the price of the car. Her ordering description may be $Od_{car} =$

$\mathit{fuel\_efficiency} : \{\mathit{manufacturer} = \texttt{USA}, \mathit{manufacturer} = \texttt{Japan} : \mathit{price}\}$. This denotes that the result should be sorted by fuel efficiency. If two cars have the same efficiency, then USA cars should come before Japanese. Finally, if two cars have the same fuel efficiency and they are both produced in Japan, then they should be ordered relative to their price. Note that in order for this order description to be valid, it must be the case that American and Japanese cars are disjoint sets and there are no other cars (i.e., all cars are manufactured either in the USA or Japan). If the last assumption does not hold, then the ordering description can be rewritten as: $Od_{car} = \mathit{fuel\_efficiency} : \{\mathit{manufacturer} = \texttt{USA}, \mathit{manufacturer} = \texttt{Japan} : \mathit{price}, \neg(\mathit{manufacturer} = \texttt{USA} \sqcap \mathit{manufacturer} = \mathit{Japan})\}$, which denote that cars that are neither American nor Japanese should come last among the cars with the same fuel efficiency.

A query is defined as a pair of a concept description and description ordering: $\langle D, Od\rangle$. An example query is $\langle Car \sqcap color = Red, Od_{car}\rangle$, which will return all things that are described as (or can be inferred to be) red cars in the order Od_{car}.

4 Index structures for Concept Descriptions

In relational databases, tree indices are used for efficiently answering range queries (e.g., give me all employees that make more than 100K) or order-by queries (e.g., give me all employee ordered by age), while hash table indices are useful for efficiently answering partial match queries (e.g., give me information about the employee with the following employee number). These indices are built based on the defined workload (i.e., queries, updates, and their frequency): see for example (Agrawal, Chaudhuri, and Narasayya 2000; Valentin et al. 2000). Here we will take the same approach and show examples of how to build custom indices for different parameterized queries. We propose directions on how to construct an index advisor for OWL-DL queries. We do not specify what type of hash tables (e.g., linear hashing (Litwin 1980) or extensible (Fagin et al. 1979) hashing) and what type of search tree data structure (e.g., AVL trees (Adelson-Velskii and Landis 1962), B+ trees (Bayer and McCreight 1972), to name a few) are to be used because these are orthogonal choices.

Hash Tables

Hash tables are useful for efficiently answering queries that do not define an ordering because the data in hash tables is spread using a hash function. We therefore consider only hash tables for queries with no ordering condition.

A general definition of a hash table for concept descriptions follows.

Definition 3 (hash table) *A hash table H has the syntax $\langle C, f\rangle$, where C is a concept description and f is a concrete feature. It partitions all concepts descriptions that are subsumed by C using a hash function on the different values of f.*

The hash table $\langle C, f\rangle$ is useful for efficiently answering queries of the form $\langle D \sqcap f = k, \epsilon\rangle$, where $\mathcal{T} \models D \sqsubseteq C$.

For example, consider a query with concept description $Car \sqcap \mathit{manufacturer} = USA \sqcap \mathit{yearBuilt} = k$ and empty ordering description. To efficiently support this query, we can create a hash table on the concrete feature *yearBuilt* that stores elements from the view that contains only concepts that are subsumed by the concept $Car \sqcap \mathit{manufacturer} = USA$, that is, that hash table $\langle Car \sqcap \mathit{manufacturer} = USA, \mathit{yearBuilt}\rangle$. We are therefore going to store in the hash table only concept description that describe cars manufactured in the USA. Next, we can create buckets based on the value of the *yearBuilt* concrete feature.

Note that a single concept may be copied to several buckets. For example, a concept in the knowledgebase may be described as $CAR \sqcap \exists \mathit{safety\_feature}.ESC$ and the terminology may contain the knowledge that American car manufactures first introduced electronic stability control in 1997. This knowledge may be represented as $CAR \sqcap \exists \mathit{safety\_feature}.ESC \sqsubseteq \mathit{yearBuild} \geq 1997$. Based on this knowledge, the input concept only needs to be copied to buckets that represent values for the built years between 1997 and 2009. Given an instance of the query and a specific value for k, the hash function will tell us the bucket that should be searched. For each concept description C in the bucket, we need to check whether $C \sqsubseteq \mathit{yearBuilt} = k$. If this is the case, then the concept description will be added to the result.

We realize that hash tables over concept descriptions can result in storage overhead. In our example, a hash table should only be created if we know that the precise value for *yearBuilt* is specified or can be derived for for most concepts in our knowledgebase. Another disadvantage is that propagating an update to a hash table can be costly because of the presence of multiple copies of the same concept description. The advantage of our approach is that it clusters the data among buckets and therefore, in almost all cases, will results in improved search performance relative to the the naïve approach of sequentially traversing through all the concepts.

Search Trees

The general definition of a search tree for concept description follows.

Definition 4 (search tree) *A search tree has the syntax $\langle C, Od\rangle$. It describes a tree that contains all concept descriptions D for which $\mathcal{T} \models D \sqsubseteq C$. If concept description D_1 comes before concept description D_2 in the search order, then it must be the case that $\mathcal{T} \not\models D_2 \prec_{Od,\mathcal{T}} D_1$.*

Note that an ordering description does not define a total order. Therefore, it may be the case that two concept descriptions are incomparable relative to an order description and therefore the order in which they appear in the search tree is not defined.

Consider our example query from Section 3: $\langle Car \sqcap color = k, \mathit{fuel\_efficiency} : \{\mathit{manufacturer} = \texttt{USA}, \mathit{manufacturer} = \texttt{Japan} : \mathit{price}\}\rangle$. We can efficiently answer this query using the search tree $\langle Car, color : \mathit{fuel\_efficiency} : \{\mathit{manufacturer} = \texttt{USA}, \mathit{manufacturer} = \texttt{Japan} : \mathit{price}\rangle$.

Next, we will describe how to search in binary trees, where searching in other tree types is analogous. Let D be the query concept description (e.g. $\langle Car \sqcap color = red\rangle$) and suppose that C is the concept description for the root node. If $D \prec_{Od,\mathcal{T}} C$, then we know that we only need to search in the left subtree. Similarly, if $C \prec_{Od,\mathcal{T}} D$, then we only need to search in the right subtree for the result. Finally, if neither of the two conditions is true, then we will have to search in both the left and right subtrees.

Note that, unlike the relational case, we are not guaranteed to return the first concept of the query result in logarithmic time and subsequent concepts in amortized constant time. The reason is that we may have to search in both the left and right subtree of the current tree node to find the concept we are searching for.

Inserting a new concept in a description tree is not a trivial operation. For example, consider a node with concept description C_1 that is $age \geq 1 \sqcap age \leq 8$ and a left child node with concept description C_2 that is $age \geq 5 \sqcap age \leq 7$. Suppose that the concepts in the tree are ordered relative to the ordering description $\langle age\rangle$. Now, if a new concept description C_3 that is $age = 3$ is inserted in the knowledgebase, then it can be inserted either in the left or right child subtree of the root node because C_1 and C_3 are not comparable relative to the defined search tree order. However, inserting C_3 in the right subtree is obviously the wrong choice because $C_3 \prec_{age,\mathcal{T}} C_2$.

One way we can solve this insertion anomaly is to always consider both the left and right subtree of the current node when performing an insertion. If at some point the query concept description is strictly smaller or bigger than the concept description in the current node, than we know that the concept should be inserted to the left or to the right, respectively, of the current node. This process can substantially slow down concept insertion and the logarithmic time-bound from the relational database case is no longer guaranteed.

Another way the described anomaly for concept insertion can be avoided is to require that there is a total order on the indexed concept descriptions. One way to insure that this is the case is to require that all indexed concept description satisfy the *descriptive sufficiency* property ((Pound et al. 2007)), which is described next.

Definition 5 (descriptive sufficiency) *A concept description D is sufficiently descriptive relative to an ordering description Od and terminology $\mathcal{T}$, written as $DS_{\mathcal{T},Od}(D)$ if at least one of the following holds.*

- $Od = \epsilon$
- $Od = f : Od_1$, $DS_{\mathcal{T},Od_1}(D)$ *and* $\mathcal{T} \models D \sqsubseteq f = k$ *for some* k
- $Od = (D_1 : Od_1, \ldots, D_m : Od_m)$ *and* $DS_{\mathcal{T},Od_i}(D)$ *and* $\mathcal{T} \models D \sqsubseteq D_i$ *for some* i

Conclusion

In this paper we have proposed a new language for querying knowledgebases and described novel hash table and search tree data structures that can be used to efficiently support this language. We realize that this proposal outlines only the first steps in building an industrial strength knowledgebase with efficient access. Future research topics include creating experimental results with the proposed hash table and search tree data structures over real-world data to measure their performance. Algorithms for automatically crating indices from a workload and query optimization techniques that rewrite queries to use the available indices are also needed. Another important research problem is creating distributed and replicated version of the proposed data structures to allow for efficient concurrent access to huge knowledgebases, even knowledgebases that will be capable of storing the whole Internet.

Acknowledgments

I would like to acknowledge past and ongoing fruitful collaboration with Jeffrey Pound, David Toman, and Grant Weddell from the University of Waterloo. This paper was inspired by collaboration with them and their valuable suggestions are included in it.

References

Adelson-Velskii, G. M., and Landis, E. M. 1962. *An Algorithm for the Organization of Information*, volume 3.

Agrawal, S.; Chaudhuri, S.; and Narasayya, V. 2000. Automated Selection of Materialized Views and Indesex for SQL Databases. *Very Large Databases (VLDB)* 495–505.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F. 2003. *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press.

Bayer, B., and McCreight, E. M. 1972. Organization and maintenance of large ordered indexes. *Acta Informatica* 1(3):173–189.

Fagin, R.; Nievergelt, J.; Pippenger, N.; and Strong, H. R. 1979. Extendible hashing - a fast access method for dynamic files. *ACM Transaction on Database Systems* 4(3):315–344.

Haarslev, V., and Möller, R. 2003. Racer: A Core Inference Engine for the Semantic Web. *Second International Workshop on Evaluation of Ontology-based Tools.*

Haarslev, V.; Möller, R.; and Wessel, M. 2004. Querying the semantic web with Racer + nRQL. *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics.*

Lawrence, S., and Giles, C. L. 1999. Accessibility of information on the web. *Nature* 400(107).

Litwin, W. 1980. Liean Hashing: a new tool for file and table addressing. *Proceedings of International Conference on Very Large Databases (VLDB)* 212–223.

Morrison, D. R. 1968. PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM* 15(4):514–534.

Web Ontology Language (OWL). *http://www.w3.org/2004/OWL.*

Pound, J.; Stanchev, L.; Toman, D.; and Weddell, G. E. 2007. On ordering descriptions in a description logic. *International Workshop on Description Logics*.

Pound, J.; Stanchev, L.; Toman, D.; and Weddell, G. E. 2008. On Ordering and Indexing Metadata for the Semantic Web. *International Workshop on Description Logics*.

Resource Description Framework (RDF). *http://www.w3.org/RDF*.

Sirin, E., and Parsia, B. 2007. SPARQL-DL: SPARQL Query for OWL-DL. *OWL: Experiences and Directions Workshop*.

Tsarkov, D., and Horrocks, I. 2006. FaCT++ Description Logic Reasoner: System Description. *International Joint Conference on Automated Reasoning (IJCAR)*.

Valentin, G.; Zulian, M.; Zilio, D.; Lohman, G.; and Skelley, L. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend its Own Indexes. *International Conference on Data Engineering* 101–110.