

ONTOLOGY-BASED FREE-FORM QUERY PROCESSING
FOR THE SEMANTIC WEB

by

Mark Vickers

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

June 21 2006

Copyright © 2006 Mark Vickers

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Mark Vickers

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

David W. Embley, Chair

Date

Deryle Lonsdale

Date

Mike Jones

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Mark Vickers in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

David W. Embley
Chair, Graduate Committee

Accepted for the Department

Parris Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean
College of Physical and Mathematical Sciences

ABSTRACT

ONTOLOGY-BASED FREE-FORM QUERY PROCESSING FOR THE SEMANTIC WEB

Mark Vickers

Department of Computer Science

Master of Science

With the onset of the semantic web, the problem of making semantic content effectively searchable for the general public emerges. Demanding an understanding of ontologies or familiarity with a new query language would likely frustrate semantic web users and prevent widespread success. Given this need, this thesis describes AskOntos, which is a system that uses extraction ontologies to convert conjunctive, free-form queries into structured queries for semantically annotated web pages. AskOntos then executes these structured queries and provides answers as tables of extracted values. In experiments conducted AskOntos was able to translate queries with a precision of 88% and a recall of 81%.

ACKNOWLEDGMENTS

I would like to express my gratitude to my family and professors who have helped make this thesis possible. I thank my wife for her patient support and love. I thank Dr. Embley for his feedback, mentoring, and for being the good man that he is. I am also grateful to all the professors at BYU who have made my experience here so enjoyable.

Contents

Acknowledgments	vi
List of Figures	ix
1 INTRODUCTION	1
2 RELATED WORK	5
2.1 Semantic Web Querying Systems	5
2.2 NLIDBs	8
3 QUERY PROCESSING	11
3.1 Introduction to Extraction Ontologies	11
3.1.1 Extracting Values from Text	13
3.1.2 Extracting Operations from Queries	18
3.2 Match Query to a Context	21
3.3 Formulate Query	24
3.4 Execute Query	28
4 EXPERIMENTAL RESULTS	33
4.1 Procedures	33
4.2 Metrics	34
4.3 Results	35
4.4 Issues	37
4.4.1 System Issues	37
4.4.2 Domain Issues	39
4.4.3 English Issues	40

5	CONCLUSION AND FUTURE WORK	41
5.1	Future Work	41
	Bibliography	46
A	Instruction Packet	47
B	Experimental Results for the Penultimate Version of AskOntos	55
C	All Third-Round Queries with Hand-Generated and System-Generated Trans- lations for AskOntos version 2	57
D	All Third-Round Queries with Hand-Generated and System-Generated Trans- lations for AskOntos version 1	67
E	All Second-Round Queries with Hand-Generated and System-Generated Trans- lations for AskOntos version 1	77

List of Figures

1.1	Web page data extraction.	3
1.2	Process-flow view of AskOntos.	4
3.1	An extraction ontology describing the concept of a car.	12
3.2	A value recognizer for car-ad prices.	14
3.3	A web page containing a car-ad record.	15
3.4	Recognized strings from in the first extraction phase.	16
3.5	Snippet from the OWL file associated with the Car Ad ontology.	19
3.6	A less-than operation recognizer for the <i>Price</i> 's data frame.	20
3.7	An extraction ontology for diamonds.	23
3.8	Generic query of the sample user query.	24
3.9	XQuery expression derived from the example generic query.	26
3.10	Records returned by XQuery engine.	29
3.11	Results transformed to HTML.	30
3.12	Capture of the web page containing a car-ad record with the extracted values highlighted.	31
4.1	Experimental results for AskOntos.	36
B.1	Experimental results from both the second and third round for the penultimate version of AskOntos.	56

Chapter 1

INTRODUCTION

With estimates of more than 11.5 billion indexable pages [15], the web is an incredibly rich source of information. The challenge of harnessing this information, making it easier to search and query, has been the focus of much research. While great strides have been made towards this goal, searching is still an application with significant room for improvement [14].

A proposed framework, known as the semantic web [2], promises to significantly enhance web querying. Whereas the current web contains information that is human readable, the semantic web extends the information to be machine-readable as well. To achieve this goal, the semantic web uses ontologies. An ontology is a formal, explicit specification of a conceptualization [13]. Ontologies will allow web search programs to look for pages containing a precise concept or answer a specific question. Semantic web search engines promise to be a great improvement over current web search engines, which typically return all pages that contain given (often ambiguous) keywords.

Even with the upcoming semantic web framework, the details of how humans are to query the semantic web are unclear. Two major issues to consider for this problem are the usability of the interface, and the effectiveness of the query processing. The user interface should require a minimal learning curve yet still allow complex queries. Queries should be processed in a way that takes advantage of semantic content on the web, aiming at interpreting a query's meaning instead of viewing it as a set of keywords. Whether this ideal can be achieved remains unclear.

This thesis introduces a system called AskOntos. AskOntos uses a novel approach to query processing that contributes to the realization of enhanced searching on the semantic web. The approach relies on extraction ontologies, which are ontologies used by an extraction engine to extract and structure domain-relevant information from unstructured web pages [7]. Extraction ontologies play a double role in the AskOntos system. First, an extraction engine uses them to extract data values from the web. With respect to an extraction ontology, the extraction engine stores extracted data values, and caches annotated versions of the processed web pages. These cached pages constitute semantic web pages, human-readable pages that have been annotated for machine processing. Second, AskOntos uses extraction ontologies for query processing. AskOntos uses an extraction engine to have each extraction ontology extract over the user’s free-form query. AskOntos chooses the extraction ontology that is responsible for extracting the most instances from the query. With the chosen ontology, and the extracted query instances, AskOntos produces a query that it executes over the previously extracted web page data values associated with the ontology.

Figure 1.1 illustrates web page data extraction. *Ontos* [28], an ontology-driven extraction engine, uses an *Extraction Ontology Repository* to parse web pages (*WPs*) and extract data values. If enough data values are extracted from a given page with respect to an ontology, the page is considered relevant to that ontology’s domain, and a semantic web page (*SWP*) is created. An *SWP* is a cached copy of a *WP* together with data that has been extracted from it with respect to an ontology. The double arrowhead lines between the *SWPs* and the extraction ontologies indicate that they reference each other.

Figure 1.2 illustrates the process flow for AskOntos. First, AskOntos passes the free-form, natural-language query to *Ontos*, which extracts recognized text strings from the query using the extraction ontologies. The *Ontology Selector* ranks the ontologies according to the number of recognized text strings they extract, and chooses the ontology that extracts more text strings than any other as the best-fit context for the query. With the chosen ontology and the recognized text strings in the query, the *Query Generator* formulates an executable query. The *Query Processor* runs the formulated query against the data values in the *SWPs* associated with the chosen ontology, producing the answer to

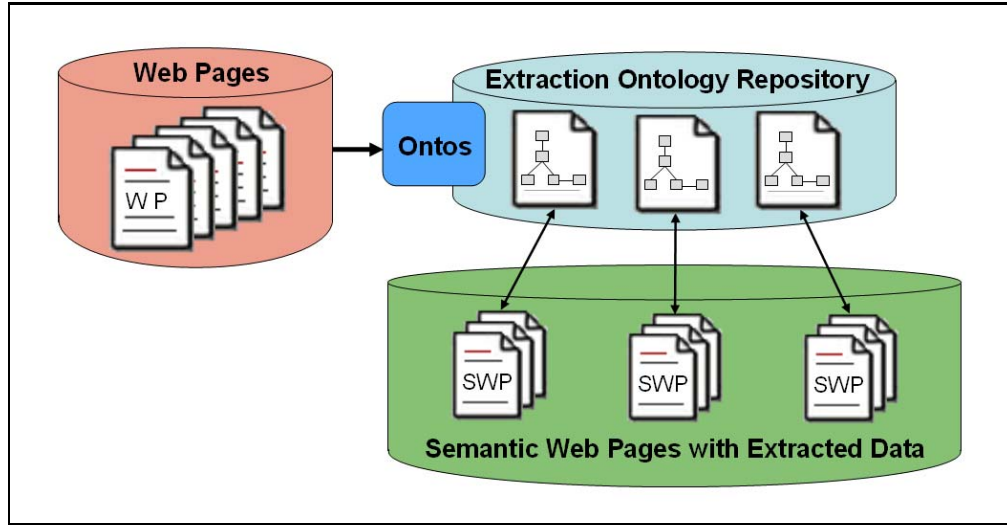


Figure 1.1: Web page data extraction.

the query. AskOntos returns database-like tables of extracted values. In addition to the extracted values, each row in the table contains a link to a cached copy of the *SWP* from which its values were extracted. When a user clicks on this link, AskOntos displays the cached page and highlights extracted data values.

Because of its use of extraction ontologies, AskOntos offers three significant benefits to semantic web query processing: 1) it assigns context to queries based on extracted values, 2) it converts free-form queries into structured queries without using any traditional natural language processing techniques such as part-of-speech recognition, and 3) it answers queries with tables of extracted values. The initial version of AskOntos can process database-like queries written as free-form, natural-language queries that may be incomplete sentences and need not be grammatically correct. Queries may also include common symbols such as $<$, $>$ and $<=$. AskOntos can also process queries requiring aggregation functions: min, max, sum, count, and avg. The system, however, currently can only process conjunctive queries—queries where atomic conditions must all hold. Other current limitations include the system’s inability to handle negations or metadata questions.

The remaining chapters of the thesis present the details of AskOntos. Chapter 2 compares AskOntos to other natural-language query processing systems. Chapter 3 first

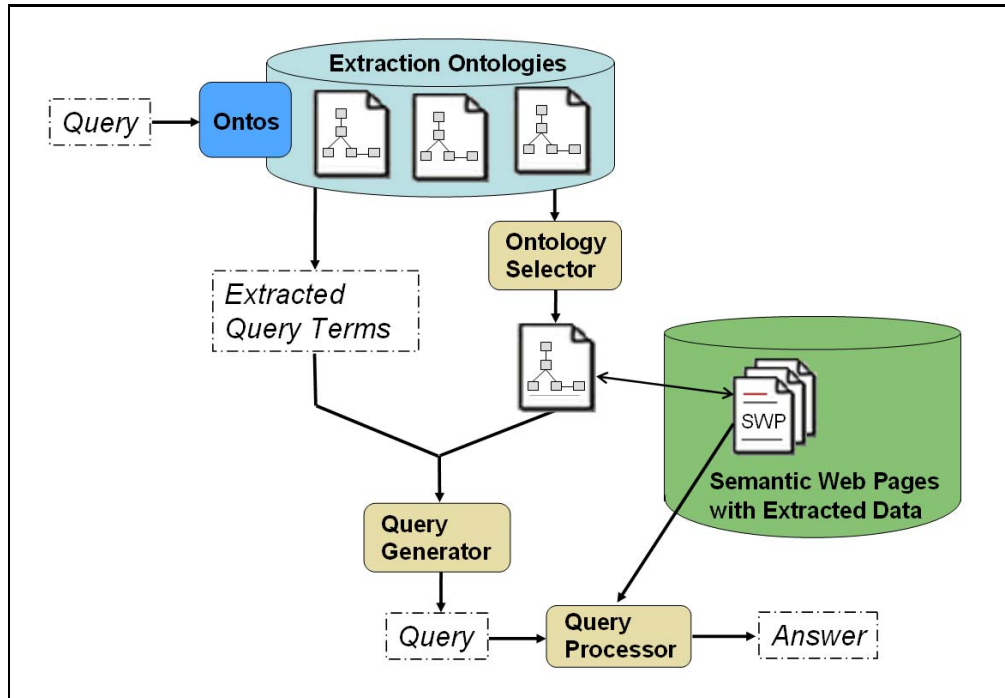


Figure 1.2: Process-flow view of AskOntos.

explains how extraction ontologies work and then describes the details of AskOntos. Chapter 4 discusses experimental results. Chapter 5 concludes the thesis with a discussion of contributions and proposed future work.

Chapter 2

RELATED WORK

Several other research efforts are similar to AskOntos in that they process queries over semantic web pages. We survey these efforts in Section 2.1. Other research efforts are similar to AskOntos in that they process natural language queries over databases. We discuss these efforts in Section 2.2

2.1 Semantic Web Querying Systems

QUEST [1] is a semantic web querying system that facilitates expressing complex queries by using a graphical query language. Their interface to the semantic web consists of a semantic view (an ontology-like graph), and a visual view (HTML). QUEST users choose ontological categories and express constraints. Query results come in the form of generated documents and graphs. The graphical interface has many advantages, but requires the user to be familiar with underlying graph structures. AskOntos is similar in that it relies heavily on the structure of ontologies, but differs in that it has a free-form text interface. The free-form text interface requires no knowledge or understanding of ontology structures from the user and therefore has a smaller learning curve.

As with QUEST, the SHOE [17] approach has the user enter a query by interacting directly with ontologies. The interface is form-based rather than graph-based. The user can drill down an ontology structure and set constraints through pull-down menus and text areas. The SHOE approach searches only web pages annotated by SHOE, but allows the users to optionally submit keyword queries to a popular information retrieval style search engine. While SHOE shares AskOntos's ability to return answers in tabular form, their interfaces differ. SHOE does not permit free-form textual queries.

The authors of [3] share the idea of making a natural language front end for semantic web queries. The natural language they propose, however, is limited to a subset of English called Attempto Controlled English (ACE) [10], which has a domain-specific vocabulary and a restricted grammar in the form of a small set of construction and interpretation rules. The system translates ACE queries into discourse representation structures (DRS) [18]. DRS terms match against ontology keywords and relations to form a process query language (PQL) statement [19], which queries an ontology. While using ACE helps overcome some major natural language processing pitfalls, it does require the user to learn the rules of the ACE language. They report that learning a controlled language takes a couple of days for the basics and 4-6 weeks for full proficiency.

AQUA [26] is an ontology-driven question answering system with a natural language interface that integrates computational linguistics, logic, question classification, and information retrieval. AQUA first tries to translate the user query into a logic form in order to do a proof of the query over a knowledge base. If the proof fails, AQUA resorts to a more traditional question answering approach to satisfy the query. To translate the user query into its logic form, AQUA first parses the query into its grammatical components and then produces a Query Logic Language (QLL) expression. AQUA then converts the QLL expression to standard predicate logic. In this conversion, AQUA uses an ontology to instantiate type variables and allows them to be replaced by unary predicates (using exact matching between terms in the ontology and terms in the QLL expression). Finally, AQUA re-writes the logic formulae by replacing predicate names with relations in the ontology. This transformation is done by their similarity algorithm. The similarity algorithm creates a graph from the query and finds its best intersection with the ontology graph by using a hand-crafted domain dictionary and node/relation labels and returns a relation name that may replace a predicate in the logic expression. Both AQUA and AskOntos use ontologies to convert a natural user query into a formal query. One difference is that AQUA utilizes a single domain ontology (useful in a company intra-net for example), while AskOntos utilizes many ontologies and is designed as an interface for the semantic web. AQUA also differs from AskOntos in that AQUA segments the query into subjects, verbs, prepositional phrases, adjectives, and objects. Also, AQUA uses an ontology to replace query terms with

terms that match the ontology/knowledge base, while AskOntos uses ontologies to first find the best context for the query and then maps query terms to concepts.

A semantic web search engine (SWSE) prototype is presented in [11]. Their system has a Google-like interface that queries over RDF documents and returns a sequence of subject, predicate, object/subject strings to answer the query. Their query processing is as follows. First, the SWSE matches query words against `rdfs:label`, `rdfs:comment`, and `rdfs:Literal` elements found in both RDF domain ontologies and an OWL translation of WordNet [9]. This results in a list of property and predicate URIs, weighted according to frequency. Next, the system generates RDF queries (sets of property-predicate-property triplets) from the permutations of the property and predicate resources, including zero or one wildcard ('?') in each triplet. The wildcard allows properties or predicates that were not matched by the query, but that may be relevant, to be introduced. After the queries are executed over an RDF knowledge base, the system adds resources in the resulting RDF statements that were found through wildcard matches to the URI collection, and generates a new set of RDF queries. The SWSE repeats this process a few times, each time expanding its list of URIs. After several iterations, the resulting set of RDF statements constitute what they call a "semantic webgraph." In attempts to match multiple RDF statements to the user query, the system counts the number of user query words that match properties and then traverses that many properties of the semantic webgraph. Results are ranked before they are displayed. Both the SWSE and AskOntos share the idea of returning semantically annotated data from a knowledge base. While neither system uses part-of-speech recognition for query interpretation, they differ on how query context is determined. The SWSE system relies on the labels, comments, and literals of RDF documents on the web and WordNet for context matching. AskOntos relies on a repository of carefully designed extraction ontologies for query interpretation. In this regard, the SWSE system scales with less effort, but AskOntos can be more precise about exactly what context and value phrases should be matched by an ontology.

2.2 NLIDBs

Ever since the early sixties researchers have been working on building natural language interfaces to databases (NLIDBs) [5]. The ideal NLIDB system would appropriately interpret an unrestricted natural language query from an inexperienced user. Due to many problems and limitations in natural language parsing technology [4], state-of-the-art systems are far from meeting these lofty goals. A typical NLIDB architecture has an analyzer and a translator. Using a generalized grammar, lexicon, and domain knowledge the analyzer does syntactic and semantic processing on the input query, converting it into a logical, intermediate representation. The translator takes the intermediate representation and does task-specific and pragmatic processing, producing a database query.

While both AskOntos and NLIDB systems take a single natural language query as input and produce a formal query as output, there are some important distinctions between them. One difference between the systems lies in the initial grammar-based processing. NLIDBs use generalized grammars to do syntactic analysis, which deals with the structural form of the input, and tries (often not very successfully) to handle fragmented and ill-formed sentences. AskOntos has domain-specific grammars encoded in each extraction ontology and makes no structural analysis and therefore handles fragmented and ill-formed sentences well. As a tradeoff with its free-form tolerance, however, AskOntos currently only handles conjunctive queries and queries with aggregations, while NLIDBs can handle more complex queries. Another difference between the systems is in how new domains are introduced. Porting an NLIDB to a new domain typically requires low-level system expertise as well as the know-how to modify lexicons, grammars, and task-specific processing done in the NLIDB's translator module. While the AskOntos engine itself is domain independent, extraction ontologies must be carefully designed and created for each domain. It is not trivial to add or change domains with either system, but AskOntos appears to require less computer/system knowledge and training.

Because it is recent (showing the current state of the art) and because its back-end processes the same XML query language that AskOntos does, XQuery [29], we single out NaLIX [21] and compare it with AskOntos. NaLIX is a system that translates natural language queries into Schema-Free XQuery [20] expressions and then executes them

over an XML database. A Schema-Free XQuery expression is an XQuery expression with additional functionality that allows the expression to have some freedom by not requiring the its elements/attributes and structures to match precisely with target XML database elements/attributes and structures. In the first step of translating a natural language query into a Schema-Free XQuery, NaLIX uses MINIPAR [22], a broad-coverage natural language parser, to create a parse tree. NaLIX then classifies the parse tree nodes as either belonging to a certain XQuery component (such as a **return** clause, an **order by** clause, a function, etc.), or a non-XQuery component that contributes to the query’s semantics (such as pronouns, prepositions, adjectives, etc.). After classification, NaLIX validates the parse tree to verify that it can be translated to a Schema-Free XQuery. If not, NaLIX begins a user feedback loop. When the query can be translated, NaLIX analyzes the structure of the parse tree and the node classifications to formulate a Schema-Free XQuery. Since NaLIX and AskOntos both convert a natural text query into an XQuery, both systems are restricted to the expressiveness of XQuery. NaLIX offers more expressiveness in the sense that it structures the XQuery according to the user query, and supports nesting, disjunctions, and negations, but the queries must be sentences parsable by MINIPAR. AskOntos always uses the same two query structures, one for conjunctive queries and one for aggregation queries, but queries need not be grammatically correct—they may be telegraphic for example and may use symbols such as $<$, $>$, and $>=$. Other differences between the systems include the fact that AskOntos is designed to be a query interface for semantic web pages, while NaLIX is designed to be an interface for an XML database. Also, in terms of query “understanding,” NaLIX has the limitation of MINIPAR’s ability to build the correct parse tree (MINIPAR achieves about 88% precision and 80% recall with respect to dependency relations with the SUSANNE Corpus [22]), and the system’s ability to expand query terms to match the underlying XML element names, which it does using WordNet or a domain ontology if the XML database has one. AskOntos is only limited, in terms of query “understanding,” by the ability of the extraction ontologies to correctly recognize domain-relevant text.

Chapter 3

QUERY PROCESSING

This chapter describes the AskOntos query processing in detail. Listed below are the three major steps:

1. Match Query to a Context
2. Formulate Query
3. Execute Query

To lay the groundwork for a detailed explanation of each of these steps, Section 3.1 describes extraction ontologies, which are a fundamental component of the approach, and how they extract instances from plain text and from queries. Following the introduction to extraction ontologies is a description of the three processing steps, comprising Sections 3.2 through 3.4.

3.1 Introduction to Extraction Ontologies

An extraction ontology is a type of conceptual model capable of performing domain-specific information extraction over plain text. Extraction ontologies include object sets, relationship sets, and constraints. Figure 3.1 shows an example of an extraction ontology for the car-advertisement domain, in graphical form. The boxes are object sets and represent a collection of instances. Object sets drawn with dashed lines contain lexical instances, while object sets with solid lines contain non-lexical instances. The arrow and dot in the *Car* object set denotes that it is the primary object, which means it is the main idea being

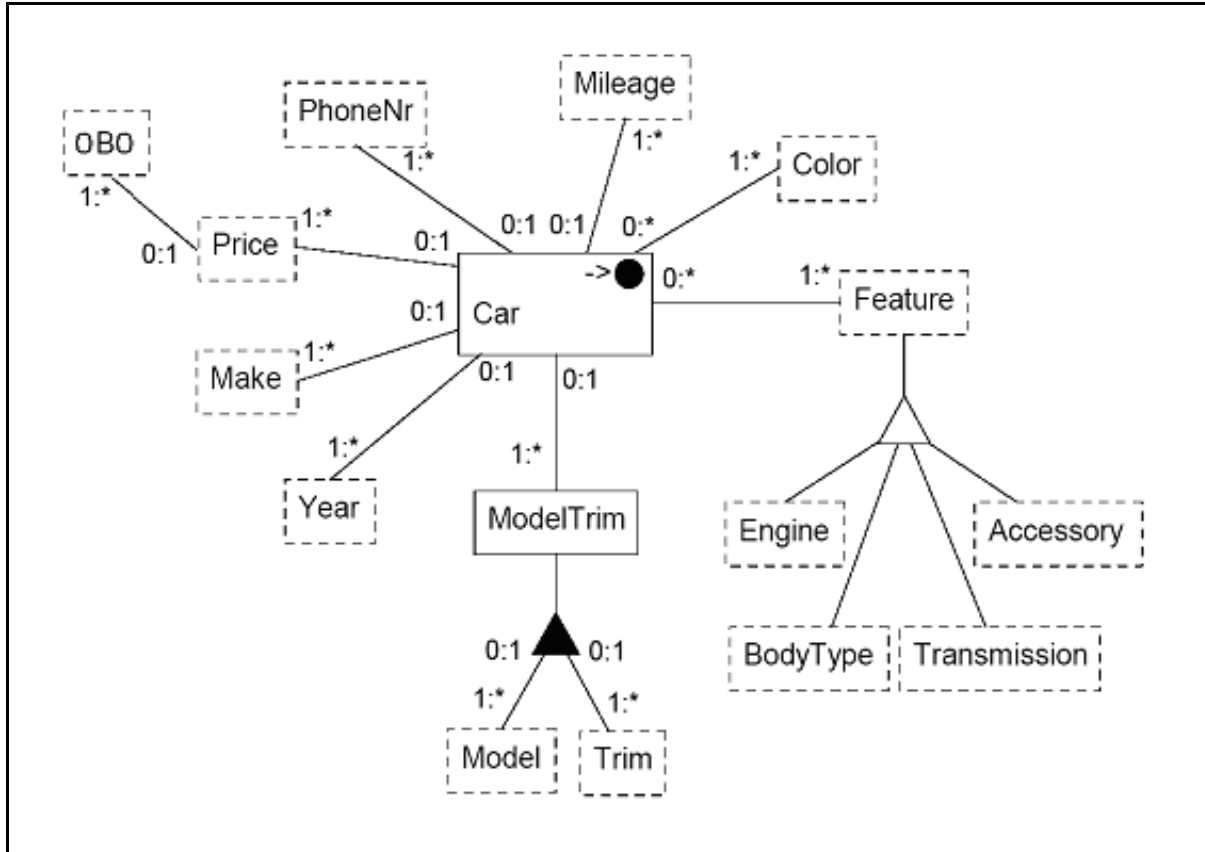


Figure 3.1: An extraction ontology describing the concept of a car.

described by the ontology. Lines between object sets are relationship sets. Numbers separated by colons, such as $0:1$, specify the minimum and maximum participation of objects in a relationship set. An asterisk (*) denotes unlimited participation. Black and clear triangles indicate aggregation and generalization/specialization respectively.

The key to extracting with extraction ontologies, and one of their distinguishing features, is that each object set has an associated data frame [6]. A data frame encapsulates the essential properties of everyday data items such as currency, dates, weights, and measures. A data frame extends an abstract data type to include not only an internal data representation and applicable operations, but also instance recognizers. Instance recognizers contain highly sophisticated representational and contextual information that allows a string that appears in a text document, statement, or query to be classified as a value belonging to the data frame or as an operator applicable to values of the data frame. Data

frames also have input canonicalization operations that convert recognized strings into a standard internal representation. Canonical values can be compared in comparison operations whereas this is not always true for strings (e.g. $48K = 48,000$ as canonical values but not as strings). Data frames also have output canonicalization operations that convert internal representations into standard display strings.

We discuss value recognition in Section 3.1.1. In Section 3.1.2, we discuss operation recognition.

3.1.1 Extracting Values from Text

The value extraction process, orchestrated by Ontos, is a two-phase process. In the first phase, Ontos takes an extraction ontology and a target text as input and applies the value recognizers (described below) of each data frame in the ontology to the text, generating a set of recognized value strings for each data frame. Because Ontos does not attempt to resolve match conflicts as part of this phase, the recognized value strings are only candidate values. In the second phase, Ontos applies several heuristics that decide for each object set which value(s) should be accepted.

Value recognizers (as well as operation recognizers discussed in the next subsection) use regular expressions to describe the textual representation of information. Figure 3.2 shows a value recognizer for the *Price* data frame in the Car Ad ontology. The value expression describes values that belong to the data frame. In this case the expression specifies whole numbers with three to six digits (we assume cars in car ads cost at least \$100) and a possible comma before the last three digits. It also specifies that the number cannot start with zero. For example, “500”, “4900”, or “12,999” are valid *Price* values. The left and right context expressions describe what must be to the immediate right and left of recognized value text strings. In this case, the left context expression indicates that a valid *Price* value must have a word boundary followed by a possible dollar sign to its left with nothing but whitespace (if any) between the dollar sign and the *Price* value. The right context must be a word boundary. The keyword expression describes words or phrases that might be near (but not necessarily immediately neighboring) recognized value text strings, acting as indicators for valid values in the extraction process. To increase expressibility, all

regular expressions in a recognizer may contain embedded lexicons. A lexicon is embedded by placing its name between curly braces in the expression. For example, the value expression for the *Color* data frame is `(light\s*|dark\s*)?\{color\}`. When Ontos performs extraction with this expression, “light ” or “dark ” may appear followed by a color named in the lexicon. A lexicon may be thought of as a regular expression with a bar (“|”) between each entry. By default Ontos processes all regular expressions as case-insensitive unless otherwise specified in the data frame.

Price Value Recognizer

value expression: `[1-9]\d{0,2},\d{3}|[1-9]\d{2,6}`

left context expression: `(\$\s*)?\b`

right context expression: `\b`

keyword expression: `(price|cost|pay|asking\s*only|obo|(or\s*)?best\s*offer)`

end

Figure 3.2: A value recognizer for car-ad prices.

To illustrate the first phase of the extraction process, consider the Car Ad ontology in Figure 3.1, including appropriate data frames for each object set, and the web page snippet in Figure 3.3. For each data frame in the ontology, Ontos applies its value recognizers to the text, producing a set of recognized value strings. Figure 3.4 shows the strings Ontos recognizes as values, keywords, and left and right contexts for each object set (only object sets with recognized strings are shown). To name a few incidental points about the extracted values, the *Make*, *Trim*, *BodyType*, and *Accessory* values are all identified by regular expressions with embedded lexicons. The *Color* value expression does not identify “blue” in “bluebook” because it does not end with a word boundary. The right context expression for *Mileage* (which is `(\s*k)?(\s*mi\.?|\s*miles)?\b`) does not recognize the “k” at the end of “bluebook” because right and left context expressions are not matched alone, but are combined with the value expression. Consequently, if the first expression

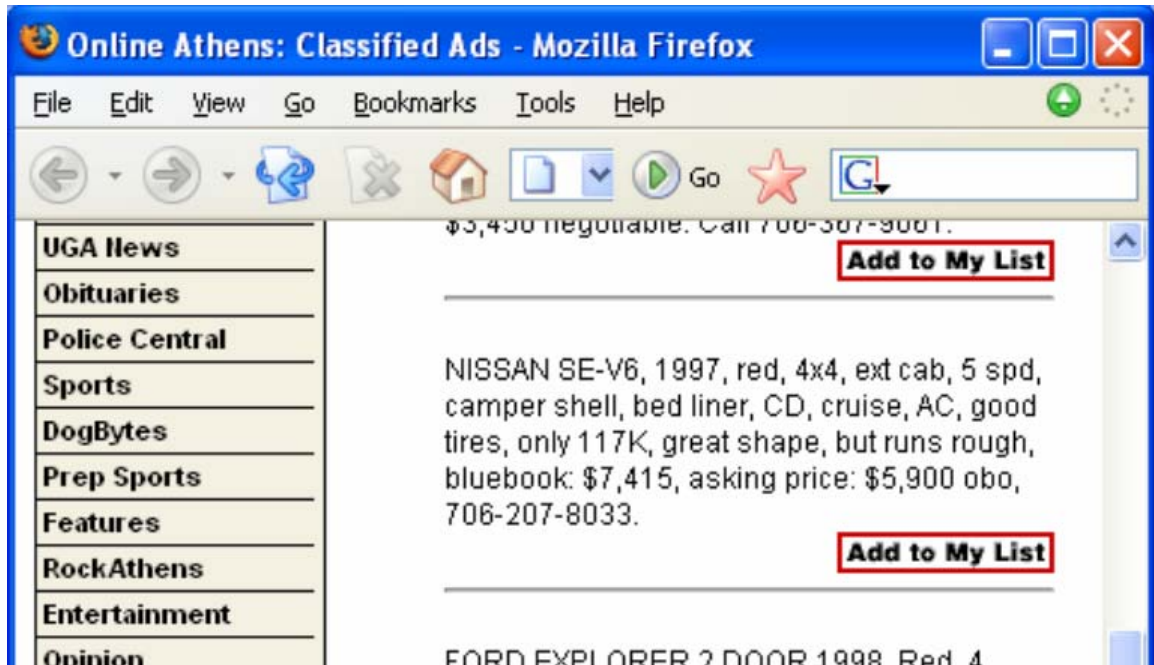


Figure 3.3: A web page containing a car-ad record.

group in the *Mileage* right context expression ($\backslash s^*k$) were not optional (did not have the ?), “117” would be the only value recognized as a *Mileage* in the given text.

In the second value extraction phase, Ontos resolves conflicting matches and decides which recognized values belong to which object sets. In many instances, this decision is trivial. For example, no value recognizers identify “NISSAN” as a value other than the *Make* value recognizer, and since “NISSAN” is the only value recognized by the *Make* value recognizer, Ontos accepts “NISSAN” as a *Make* value. Similarly, Ontos accepts “SE”, “red”, and “obo” as *Trim*, *Color*, and *OBO* (“or best offer”) values respectively. Because the participation constraints for the *Accessory* object set ($0..*$) specify an unbounded upper limit, all recognized *Accessory* value strings that do not overlap with recognized value strings of other object sets, which excludes “camper shell,” are accepted as belonging to the *Accessory* object set.

As a first attempt at sorting out conflicting matches, Ontos rejects all recognized value strings that are subsumed in the text by other recognized value strings. As a result, “camper” is rejected as a *BodyType* value and “706”, “207”, and “8033” are rejected as

Recognized Values	<u>Make</u>	<u>Trim</u>	<u>Year</u>	<u>Color</u>	<u>BodyType</u>
	NISSAN	SE	1997	red	camper
	<u>Price</u>	<u>Mileage</u>	<u>Accessory</u>	<u>OBO</u>	
	1997	1997	4x4	obo	
	117	117	ext cab		
	7,415	7,415	camper shell		
	5,900	5,900	bed liner		
Recognized Keywords	<u>PhoneNr</u>	<u>Engine</u>			
	706-207-8033	V6			
	<u>Price</u>				
	price				
	obo				
	<u>Price</u>				
	\$				
Recognized Left Context	\$				
Recognized Right Context	<u>Mileage</u>				
	K				

Figure 3.4: Recognized strings from in the first extraction phase.

Price or *Mileage* values. This allows Ontos to trivially accept “camper shell” and “706-207-8033” as *Accessory* and *PhoneNr* values respectively.

The problem remains of deciding whether “1997” belongs to *Year*, *Mileage*, or *Price*, as well as deciding which, if any, of the values “117”, “7,415”, and “5,900” belong to *Price* or *Mileage*. To solve these problems, Ontos ranks the object sets according to precedence of claiming a value. The first ranking heuristic gives precedence to object sets whose right or left context expressions recognize text to the immediate right or left of a recognized value. This ranks *Price* (because of the “\$”) and *Mileage* (because of the “K”) over *Year*. To distinguish between *Price* and *Mileage*, a second heuristic favors object sets whose keyword expression identifies a keyword phrase in the text. According to this heuristic, *Price* ranks higher than *Mileage* because *Mileage* has no keyword matches and *Price* has one, “price”. Once object sets are ranked, Ontos considers each object set in turn

and accepts as many values as the object set's participation constraints allow, which happen to be at most one value for *Price*, *Mileage*, and *Year*. In deciding which value to accept as a *Price* value, Ontos rejects "1997" and "117" because they do not have recognized right or left context strings to their immediate right or left, whereas "7,415" and "5,900" do (""). Ontos accepts "5,900" and rejects "7,415" as a *Price* value because "5,900" is closer in the text to the recognized *Price* keyword, "price". Of the remaining possible *Mileage* values, "1997", "117", and "7,415", Ontos accepts "117" as a *Mileage* value because "117" is the only recognized *Mileage* value with a recognized right or left context string as a neighbor ("K"). Finally, because "1997" has not been claimed by any other object set, Ontos accepts it as a *Year* value.

In order to make the accepted values comparable with other extracted values, Ontos canonicalizes each one. This is necessary so that, for example, the extracted *Mileage* value, "117" (which is really "117,000" because of the "K") can be properly compared to other mileage values not expressed in thousands. In order to canonicalize each accepted value, Ontos calls the canonicalization operation specified, if any, in the value's associated data frame. A canonicalization operation accepts an extracted value string as an argument and returns the value string in a canonicalized form. Besides the extracted value string, canonicalization operations allow left and right context strings as arguments, which might indicate the need for special processing. For example, if the right context parameter is "K", the *Mileage* canonicalization operation not only removes any commas that might exist, but it adds three zeros to the end of the value. So, "117" is canonicalized to "117000" by the *Mileage* canonicalization operation.

In order to conveniently display an extracted value in a user-friendly format, Ontos also calls an output formatting operation, if one exists, specified in the data frame of each accepted value. The *Mileage* output formatting operation, for example, accepts a canonicalized *Mileage* value as an argument and returns the value after adding appropriate commas and appending "miles" to the end. So, "117000" becomes "117,000 miles" by the *Mileage* output formatting operation.

Ontos generates two documents as a result of performing extraction, a cached copy of the target page (typically a web page), and a document containing the extracted data.

Cached web pages are annotated with HTML tags that identify the location of each record within the page. (Records are separated as part of the extraction process by the VSM algorithm described in [27].) The document containing extracted data is an OWL (Web Ontology Language) [23] file. Figure 3.5 shows a single record from the above example saved in an OWL file. A single record consists of an instance element, one or more value elements, and an `owl:Thing` element. The instance element indicates the instance number of its record, which is unique within the OWL document. The instance element at the top of Figure 3.5 indicates that the record is the seventh instance. This instance number is appended to all `rdf:ID` attributes within the record. Ontos creates value elements from accepted values and names them after the object set to which the value belongs. Each value element contains four child elements: one for the value in its canonicalized form, one for the value in its display form, one for the offset in the cached copy of the web page from which the value was extracted, and one for the length of the originally extracted value. `owl:Thing` elements also group values together by referencing each value element in the record. The `owl:Thing` in Figure 3.5 shows that Ontos extracted *Price*, *PhoneNr*, *Mileage*, *Color*, *Make*, *Year*, *Trim*, *Engine*, *Accessory*, and *OBO* values for the seventh record. The `ontos:URI` element stores the URI to the cached copy of the web page from which the record comes, and includes the fragment (`#CarAdRecord0007`) that references the HTML tag placed in the cached copy of the web page to identify the record's location within the page.

3.1.2 Extracting Operations from Queries

Extracting information from queries is almost equivalent to extracting information from plain text. One minor difference is that for queries, record separation is not necessary, whereas it might be necessary when extracting from documents. One significant difference is that operation phrases (such as “more than” or “or greater”) are not of interest when extracting data from web pages. Since AskOntos processes queries for the purpose of converting them into formal queries, however, it is critical to properly recognize operations. As an example, each of “under”, “less than”, or “cheaper than” should trigger a *less-than*

```

<Car rdf:ID="CarInstance7">
  <CarValue rdf:datatype="xsd:string">7</CarValue>
</Car>

<Price rdf:ID="PriceInstance7">
  <canonicalValue rdf:datatype="xsd:string">5900</canonicalValue>
  <displayValue rdf:datatype="xsd:string">$5,900</displayValue>
  <cacheOffset rdf:datatype="xsd:nonNegativeInteger">14186</cacheOffset>
  <length rdf:datatype="xsd:nonNegativeInteger">5</length>
</Price>

<PhoneNr rdf:ID="PhoneNrInstance7">
  <canonicalValue rdf:datatype="xsd:string">706-207-8033</canonicalValue>
  <displayValue rdf:datatype="xsd:string">(706) 207-8033</displayValue>
  <cacheOffset rdf:datatype="xsd:nonNegativeInteger">14192</cacheOffset>
  <length rdf:datatype="xsd:nonNegativeInteger">12</length>
</PhoneNr>

<Mileage rdf:ID="MileageInstance7">
  <canonicalValue rdf:datatype="xsd:string">117000</canonicalValue>
  <displayValue rdf:datatype="xsd:string">117,000 miles</displayValue>
  <cacheOffset rdf:datatype="xsd:nonNegativeInteger">14139</cacheOffset>
  <length rdf:datatype="xsd:nonNegativeInteger">3</length>
</Mileage>

...

<owl:Thing rdf:about="#CarInstance7">
  <hasPrice rdf:resource="#PriceInstance7" />
  <hasPhoneNr rdf:resource="#PhoneNrInstance7" />
  <hasMileage rdf:resource="#MileageInstance7" />
  <hasColor rdf:resource="#ColorInstance7" />
  <hasMake rdf:resource="#MakeInstance7" />
  <hasYear rdf:resource="#YearInstance7" />
  <hasTrim rdf:resource="#TrimInstance7" />
  <hasAccessory rdf:resource="#AccessoryInstance7" />
  <hasAccessory rdf:resource="#AccessoryInstance7" />
  ...
  <hasOBO rdf:resource="#OBOInstance7" />
  <ontos:URI rdf:datatype="xsd:string">
    file:///c:/...OnlineAthensClassifiedAdds.html#record_7
  </ontos:URI>
</owl:Thing>

```

Figure 3.5: Snippet from the OWL file associated with the Car Ad ontology.

```

Less-than Operation Recognizer
  keyword expression:
    (((less(\s*expensive)?|che[ea]per|lower)\s*than)|below|under|<)\s*{Price}
  operator syntax: <
end

```

Figure 3.6: A less-than operation recognizer for the *Price*’s data frame.

operation if it were followed by a *Price* value. For this purpose, data frames may have operation recognizers in addition to value recognizers.

Operation recognizers help AskOntos translate a recognized operation phrase into a formal query constraint by recognizing the operation in the query, recognizing operands specified in the query, and specifying the appropriate operator syntax for the formal query. Figure 3.6 shows an example of a *less-than* operation recognizer for the *Price* data frame. Keyword expressions recognize operands as well as operators in queries. Object set names between curly braces in keyword expressions specify the presence of an operand. Ontos recognizes operands by replacing the object set name and curly braces with the left context, value, and right context expressions of the specified object set. Ontos separates each of these three parts into regular expression groups (places them between“()”) in order to distinguish the value from its context. For example, Ontos replaces {Price} with ((\\$ \s*)? \b)([1-9]\d{0,2}, \d{3})[1-9]\d{2,6})(\b). Text matched by the value expression group (the second group in this example) is considered to be an operand for the recognized operation. As an example, if Ontos applies the keyword expression in Figure 3.6 to “under \$6,000”, it recognizes the presence of a *less-than* operation with “6,000” as its second operand. The first operand is always the object set name, which is *Price* in this case. From this, AskOntos creates the condition: *Price* < 6000. The *less-than* symbol comes from the operator syntax specified in the *less-than* operation recognizer.

AskOntos also handles more complicated operations. For example, the *Price* data frame has a *between* operation recognizer with several keyword expressions, one of which

is between $\{Price\} \backslash s^*(and|to|-) \backslash s^*\{Price\}$. The operator syntax of the recognizer is “>, <”. If Ontos applies this operation recognizer to “between \$2,000 and \$8,000”, it extracts “2,000” and “8,000” as operands for the *between* operation. AskOntos recognizes that there are two extracted operands and two operators specified in the operator syntax (“>”, and “<”), so it creates two binary conditions for the formal query. In the first condition, *Price* is the first operand, “>” is the operator, and 2000 is the canonicalized second operand. In the second condition, *Price* is the first operand, “<” is the operator, and 8000 is the canonicalized second operand. Whenever there are two operators specified in the operator syntax of an operation recognizer, the first operator is associated with the lesser of the two operands. This allows AskOntos to correctly translate phrases such as “between \$8,000 and \$2,000” by simply ordering the operands from least to greatest, and then associating the operator syntax operators to the operands in their specified order.

3.2 Match Query to a Context

Query processing begins when the user submits a free-form query to AskOntos. AskOntos performs the first processing step by passing the query to Ontos. Ontos parses the query using each of the extraction ontologies in the extraction ontology repository.

To illustrate this process, suppose the user enters the query:

“Find me the price and mileage of all red Nissans - I want a 1996 or newer.”

With respect to the Car Ad extraction ontology, Ontos extracts “price” and “mileage” as keywords for the *Price* and *Mileage* data frames respectively. Ontos extracts “red”, “Nissan”, and “1996” as values for the *Color*, *Make*, and *Year* data frames respectively. Finally, Ontos extracts the phrase “1996 or newer” as a *greater-than-or-equal* operation in the *Year* data frame, and identifies “1996” as a parameter.

Given the information extracted from the query for each ontology, AskOntos finds the ontology in the repository (if any) that best serves as a context for the query. If the ontology repository had many domains, this could potentially be a difficult problem; however, the experimental repository only has five domains. This being the case, the ontology that best corresponds with the query is simply the one that extracts the most value and keyword

phrases from the query (although there are some special circumstances, described below). The number of matched value and keyword phrases is the ontology's *similarity value*. Operation matches are not counted towards the similarity value because many operations (e.g. less-than and greater-than) appear in most ontologies and thus provide no discriminating information. An ontology with many comparison operations may falsely accumulate a high similarity value.

To illustrate the ontology selection process, consider the Car Ad extraction ontology in Figure 3.1 and the example query in Section 3.2. From the parsing process described in Section 3.2, the Car Ad ontology has a total of five value or keyword matches: “price”, “mileage”, “red”, “nissan”, and “1996”. The similarity value for the Car Ad ontology, in this case, is 5. Now consider the Diamond ontology in Figure 3.7. The word “price” matches the *Price* keyword recognizer, and “red” matches the *Color* value recognizer (though rare, red diamonds do exist), so the Diamond ontology's similarity value is 2. Since the Car Ad ontology has a higher similarity value, it corresponds better to the query than the Diamond ontology.

Regarding similarity values, there are three cases to consider: 1) there is an obvious best match, 2) there are multiple scores that are above a minimum threshold, but none is the obvious best, and 3) all scores are below a minimum threshold. For the first case, the system automatically selects the obvious best match, which is where a single ontology has a similarity value that is α standard deviations above the mean of all similarity values. Empirical results show that $\alpha = 1.4$ works well for the testing environment. For the second case, AskOntos prompts the user to choose between the ontologies that have a similarity value above the minimum threshold. This case also includes the situation when two or more ontologies are α standard deviations above the mean, but they all have the same similarity value. For the third case, AskOntos notifies the user that the query does not match any domain. Since queries can often have low similarity values (for example the query “What is the cheapest Ford?” has a similarity value of 1 for the Car Ad ontology), the minimum threshold is currently set to 1.

In the case where the ontology repository is large, research reported in [8] may apply. This research decides if an HTML document contains objects of interest with respect

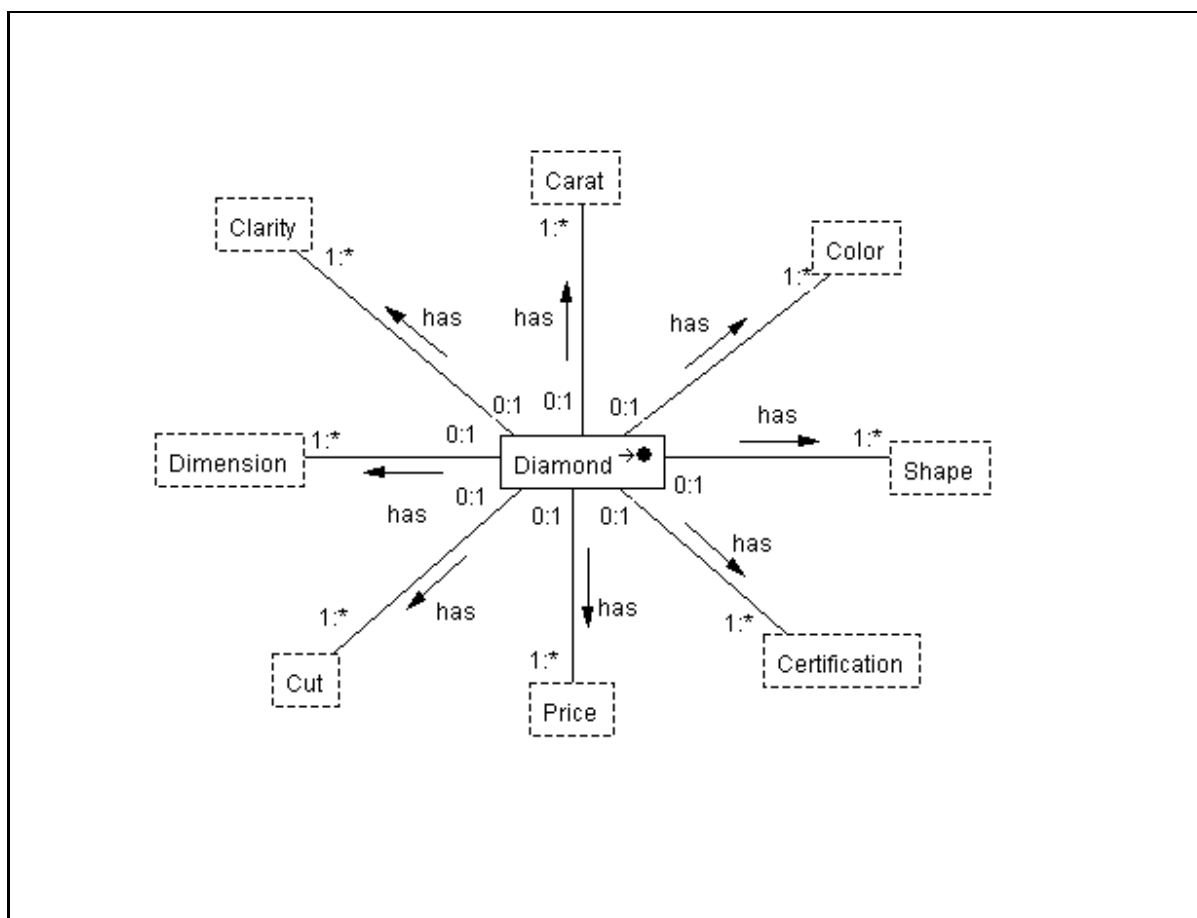


Figure 3.7: An extraction ontology for diamonds.

Return-Clause Names: Price, Mileage, Color, Make, Year
Conditions: (Color, =, red), (Make, =, nissan), (Year, >=, 1996)
Aggregations:

Figure 3.8: Generic query of the sample user query.

to an extraction ontology. The system uses machine-learned rules over density, expected-value, and grouping heuristics to decide the relevance of a document. We should be able to use some of these same techniques or similar techniques to decide how applicable a query is to an ontology.

3.3 Formulate Query

To formulate a formal query, the query generator creates a generic query from the chosen ontology's extracted phrases, and then AskOntos translates the generic query into an XQuery expression. The expression restricts records to those that do not violate any constraint, and it may make use of the XQuery's aggregation functions: min, max, sum, avg, and count.

A generic query has three list structures. The first list contains names of the object sets that belong in a return clause. The second list contains the query conditions—a condition is an attribute/operation/value triplet. The third list contains aggregations. An aggregation is a function-name/parameter pair, where the parameter is the name of an object set; for example, max/Price. Figure 3.8 shows the generic query generated from the example query in Section 3.2. Incidentally, observe that AskOntos generates conditions using the canonicalized form of each value. This ensures that generated comparison operations will work properly.

The query generator creates the three-component structure of a generic query by analyzing the matches of the various recognizers. The generator adds the names of all object sets whose data frames recognize at least one phrase (either a value, keyword or

operation phrase) to the return-clause name list. Figure 3.8 shows that returned values for the sample query are to come from the *Price*, *Mileage*, *Color*, *Make*, and *Year* object sets. The query generator creates one condition for each extracted value. The condition's attribute is the name of the object set to which the value belongs. The condition's value is the canonicalized form of the value, and by default the condition's operation is equality. If the recognized value is also a parameter for a recognized operation phrase, the query generator uses the operation syntax specified in the operation recognizer as the condition's operation. Figure 3.8 shows that since "red", "nissan", and "1996" are recognized values from the sample query, they become part of the generic query conditions. Further, because "1996 or newer" is a recognized operation, the *Year* condition is ">=". The query generator creates an aggregation for any recognized operation phrase whose recognizer is for an aggregation operation, one of *max*, *min*, *sum*, *avg*, or *count*. The parameter for an aggregation operation is the name of the object set to which the recognized operation phrase belongs. Figure 3.8 indicates that there are no aggregation operations for the sample query.

AskOntos formulates an XQuery expression from the generic query created by the query generator. For the generic query in Figure 3.8, for example, AskOntos generates the XQuery in Figure 3.9. Generated XQuery expressions follow the standard FLWOR (For, Let, Where, Order By, Return) pattern.

AskOntos generates the **For** clause as follows. Since OWL files contain an `rdf:RDF` element with child `owl:Thing` elements (which are essentially records, see Figure 3.5), one **for** loop is generated to loop over the `rdf:RDF` element(s) along with a nested loop to iterate over the `owl:Thing` elements (see Lines 1-2 of Figure 3.9).

The generated **Let** clause is a series of XQuery statements (Lines 4-25 in Figure 3.9) that produce four variables for each object set that matches any query word. Lines 5-8 show the four *Price* variables. The first of the four assigns the canonicalized *Price* value to the variable `$Price`, the second assigns the display value to `$PriceDisplay`, the third assigns the offset of the *Price* value in a cached web page to `$PriceOffset`, and the fourth assigns the length of that cached value to `$PriceLength`. The **let** statements use the `$id` variable on Line 4 to assure that all values are from the same record. The **let** statement in Line 25 assigns the URI of a cached web page to the value `$Source`. To illustrate how the **let**

```

1: for $doc in document("file:///c:/ontos/owlLib/CarAd.OWL")/rdf:RDF
2: for $Record in $doc/owl:Thing
3:
4: let $id := substring-after(xs:string($Record/@rdf:about), "CarInstance")
5: let $Price := $doc/car:Price[@rdf:ID=
      concat("PriceInstance", $id)]/car:canonicalValue/text()
6: let $PriceDisplay := $doc/car:Price[@rdf:ID=
      concat("PriceInstance", $id)]/car:displayValue/text()
7: let $PriceOffset := $doc/car:Price[@rdf:ID=
      concat("PriceInstance", $id)]/car:cacheOffset/text()
8: let $PriceLength := $doc/car:Price[@rdf:ID=
      concat("PriceInstance", $id)]/car:length/text()
9: let $Mileage := $doc/car:Mileage[@rdf:ID=
      concat("MileageInstance", $id)]/car:canonicalValue/text()
10: let $MileageDisplay := $doc/car:Mileage[@rdf:ID=
      concat("MileageInstance", $id)]/car:displayValue/text()
11: let $MileageOffset := $doc/car:Mileage[@rdf:ID=
      concat("MileageInstance", $id)]/car:cacheOffset/text()
12: let $MileageLength := $doc/car:Mileage[@rdf:ID=
      concat("MileageInstance", $id)]/car:length/text()
...
25: let $Source := $Record/ontos:URI/text()
26:
27: where($Color="red" or empty($Color)) and
28:      ($Make="nissan" or empty($Make)) and
29:      ($Year>=1996 or empty($Year))
30: return <Record ID="{ $id }">
31:   <Price>
32:     <displayValue>{$PriceDisplay}</displayValue>
33:     <cacheOffset offset="{ $PriceOffset }" length="{ $PriceLength }" />
34:   </Price>
35:   <Mileage>
36:     <displayValue>{$MileageDisplay}</displayValue>
37:     <cacheOffset offset="{ $MileageOffset }" length="{ $MileageLength }" />
38:   </Mileage>
39:   <Color>
40:     <displayValue>{$ColorDisplay}</displayValue>
41:     <cacheOffset offset="{ $ColorOffset }" length="{ $ColorLength }" />
42:   </Color>
43:   <Make>
44:     <displayValue>{$MakeDisplay}</displayValue>
45:     <cacheOffset offset="{ $MakeOffset }" length="{ $MakeLength }" />
46:   </Make>
47:   <Year>
48:     <displayValue>{$YearDisplay}</displayValue>
49:     <cacheOffset offset="{ $YearOffset }" length="{ $YearLength }" />
50:   </Year>
51:   <Source>{$Source}</Source>
52: </Record>

```

Figure 3.9: XQuery expression derived from the example generic query.

statements are processed, consider Lines 4 and 5 in Figure 3.9. The variable \$id receives the instance number of the owl:Thing element in the current iteration by taking the substring after the “CarInstance” part of the rdf:about attribute. In Line 5, \$Price receives the text found in the car:canonicalValue element whose parent element is named car:Price and has an rdf:ID attribute equal to “PriceInstance” concatenated with the instance number in the variable \$id.

AskOntos forms the **Where** clause (Lines 27-29 in Figure 3.9) using the canonicalized variable of each object set listed in the generic query’s list of conditions. Each **Where** clause has an **or empty** condition appended to it to prevent false negatives. Most current systems that retrieve information try to minimize false negatives at the expense of false positives [16]. To illustrate this principle, consider the query submitted by a user looking for red cars. If a car record does not include a color, it would be a false negative if the car was red but the record not returned. It would be a false positive if the car were not red and yet returned. The **or empty** allows the user to see possibly desired records that would have otherwise been missed.

AskOntos does not produce an **Order By** clause. Instead, AskOntos orders the resulting records after the XQuery has been executed according to the number of attribute values that are non-empty. Records with the least number of empty attribute values are displayed first.

The **Price**, **Mileage**, **Color**, **Make**, and **Year** elements in the **Return** clause (Lines 30-52 in Figure 3.9) come from the list of object sets to return in the generic query. In the return statement, each object set element has child elements with the value in its output format and the cached value’s offset and length, referenced by the appropriate variables. AskOntos returns values for all object sets referenced in the query, as opposed to just object sets that recognize keyword phrases, because it gives the user a more complete view of each record. Further this choice helps the user to know with greater certainty that the constraints specified were executed.

3.4 Execute Query

Using Qexo 1.7, a GNU implementation of an XQuery engine for Java, AskOntos runs the generated XQuery expression over the extracted data associated with the selected ontology. Qexo returns XML as specified by the query's return statement. Figure 3.10 shows the XML output from running the XQuery expression in Figure 3.9 on an OWL file with extracted data shown in Figure 3.5.

In order to display Qexo output in a more reader-friendly manner, AskOntos translates the XML results into the HTML table shown in Figure 3.11. AskOntos parses the XML result and creates a list of record structures, one for each **Record** element. Each record structure stores the attribute names (from the child element names), the **display-Value** text, and the offsets and lengths (found in the **cacheOffset** elements) within its associated **Record** element. Before printing the records as an HTML table, AskOntos sorts the records according to their number of display values.

It may be the case that a single record contains multiple values for one attribute. For example, had the user mentioned the word "accessories" in the query, a keyword match would have triggered *Accessory* values to be returned. If it were the case that a record contained multiple accessory values (which would be very likely), rather than listing them all in single table cell, AskOntos would generate an HTML button. Clicking the button would expand the cell to show all accessory values.

In the generated HTML table, each row contains a link that points to a cached copy of the page from which the extracted record values come. Clicking the link opens the cached page in a browser, and since the link contains the fragment that points to an HTML record marker in the page, the browser is scrolled to the section where the record was extracted. AskOntos uses the offset and length values in Qexo's output XML to highlight the extracted values from the row in order to help the user easily find them. Figure 3.12 shows the page displayed as a result of clicking on the first returned record in the example.

```

<QueryResult>
  <Record ID="7">
    <Price>
      <displayValue>$5,900</displayValue>
      <cacheOffset offset="14186" length="5" />
    </Price>
    <Mileage>
      <displayValue>117,000 miles</displayValue>
      <cacheOffset offset="14139" length="3" />
    </Mileage>
    <Make>
      <displayValue>Nissan</displayValue>
      <cacheOffset offset="14035" length="6" />
    </Make>
    <Year>
      <displayValue>1997</displayValue>
      <cacheOffset offset="14049" length="4" />
    </Year>
    <Color>
      <displayValue>red</displayValue>
      <cacheOffset offset="14055" length="3" />
    </Color>
    <Source>
      f:///c:/...Online&thensClassified&ads.html#record_7
    </Source>
  </Record>
  <Record ID="16">
    <Price>
      <displayValue>$24,595</displayValue>
      <cacheOffset offset="42471" length="6" />
    </Price>
    <Mileage>
      <displayValue>13,000 miles</displayValue>
      <cacheOffset offset="42440" length="2" />
    </Mileage>
    <Make>
      <displayValue>Nissan</displayValue>
      <cacheOffset offset="42385" length="6"/>
    </Make>
    <Year>
      <displayValue>2005</displayValue>
      <cacheOffset offset="42393" length="4" />
    </Year>
    <Color>
      <displayValue>red</displayValue>
      <cacheOffset offset="42402" length="3" />
    </Color>
    <Source>
      f:///c:/...SLCWeekly.html#record_16
    </Source>
  </Record>
  ...
</QueryResult>

```

Figure 3.10: Records returned by XQuery engine.

The screenshot shows a Mozilla Firefox browser window with a menu bar (File, Edit, View, Go, Bookmarks, Tools, Help) and a status bar (Done). The main content area displays a table with the following data:

Price	Mileage	Make	Year	Color	Source
\$5,900	117,000 miles	Nissan	1997	red	Online.AthensClassifiedAds.html
\$11,500	17,000 miles	Nissan	2002	red	Online.AthensClassifiedAds.html
\$9,900	670 miles	Nissan	2004		SLCWeekly.html
\$9,400	63,000 miles	Nissan	2001		Online.AthensClassifiedAds.html
\$9,900	670 miles	Nissan	2004		SLCWeekly.html
\$8,000	74,000 miles	Nissan	1997		SLCWeekly.html
\$4,100		Nissan	1996		Online.AthensClassifiedAds.html

Figure 3.11: Results transformed to HTML.

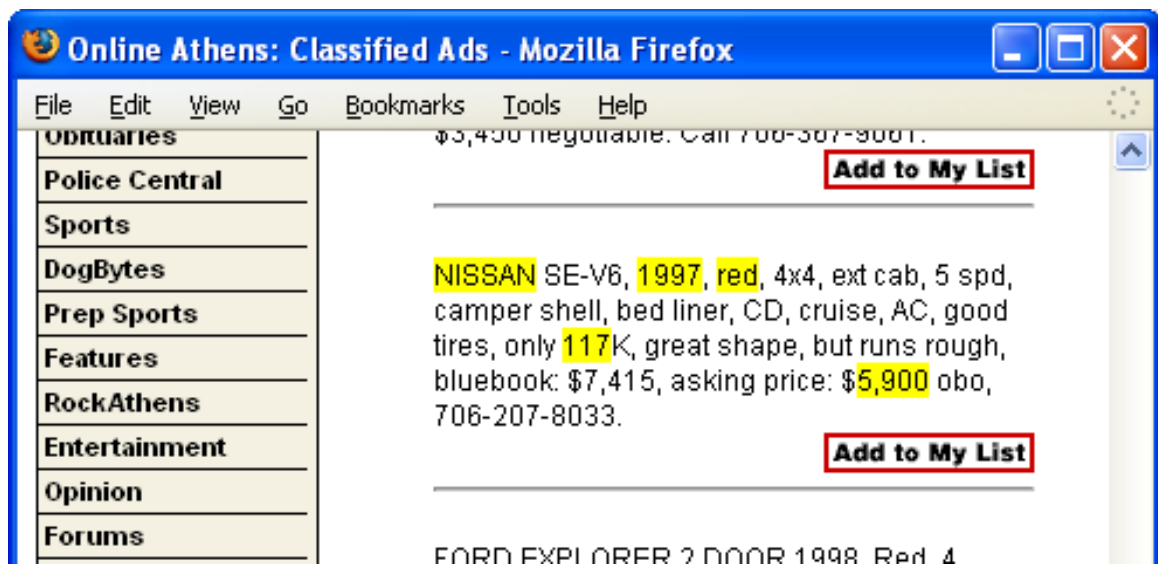


Figure 3.12: Capture of the web page containing a car-ad record with the extracted values highlighted.

Chapter 4

EXPERIMENTAL RESULTS

AskOntos has two measurable processes: the extraction process and the query translation process. Although some improvements were made to Ontos’s extraction heuristics, we did not explicitly test the performance of the extraction process. The originators of Ontos give basic performance results for the extraction process on web pages [27, 28]. Although not explicitly tested, implicit testing of the extraction process is part of testing the query translation process—poor extraction results in poor query translation. In our experiments we focused only on AskOntos’s ability to translate a user query into a formal query.

4.1 Procedures

We performed experiments using an extraction ontology repository with five domains: car ads, house ads, countries, movies, and diamonds. Subjects were computer science graduate students at Brigham Young University. None were members of our data-extraction research group. We asked subjects to submit 25 queries (5 for each domain). By way of instruction, we gave the subjects sample web-page printouts from each of the five domains and asked them to write English queries against the data in the sample web pages. The instructions also explain that the queries cannot have disjunctions or negations, cannot be metadata queries, and cannot require knowledge outside the information on the sample (or similar) web pages. Appendix A contains a copy of the instructions given to each subject. The subjects were not aware of the AskOntos process flow or underlying structures; they were only aware that it translates natural-language, free-form queries into machine-processable queries.

The experiments consisted of three rounds of testing. In the first round, subjects submitted 50 queries, but the queries did not hold to the limitations specified and were therefore discarded. A revision of the instructions to those in Appendix A solved this problem. In the second round, different subjects submitted another 50 queries. We used these queries to identify some problems and make minor adjustments to lexicons and regular expressions. In the third round, subjects (different from those in either of the first two rounds) submitted another 50 queries. The results were only slightly better than those from the second round, leading us to believe that AskOntos had become reasonably uniform in its performance. Our results reported here are from this third round of submissions for the final version of AskOntos. Appendix B gives all results from both the second and third rounds for the penultimate version of AskOntos.

4.2 Metrics

To measure AskOntos’s ability to translate natural-language, free-form queries into formal queries, we manually translated each submitted test query into the intermediate form explained in Section 3.3. We combined the aggregation list with the conditions list, however, because there was only one test query that required an aggregation operation. (One test case does not provide enough data for a meaningful result). As an example, if a query asks for the least expensive car, the condition (Price, =, min) is added to the condition list. As before, the query “Find me the price and mileage of all red Nissans - I want a 1998 or newer” is translated to

Return-Clause Names: Price, Mileage, Color, Make, Year

Conditions: (Color, =, red), (Make, =, nissan), (Year, \geq , 1998).

Since AskOntos automatically converts natural-language, free-form queries into this same intermediate form, we were able to compare each hand-written intermediate query to its generated intermediate query. Appendices C, D, and E contains all second- and third-round users’ queries and their hand-generated and system-generated translations.

The evaluation metrics we use are similar to those used in [24], which are also the standard metrics for SENSEVAL-3’s Logic Forms task [25]. As with their metrics,

we compute precision and recall over an entire set of test queries. Rather than measuring precision and recall for correctly translated arguments and predicates, however, we measure precision and recall for correctly translated return-clause names and conditions.

We calculate precision for the return-clause names as the number of correctly generated return-clause names divided by the total number of generated return-clause names. We calculate the recall for the return-clause names as the number of correctly generated return clause names divided by the number of return-clause names that should have been generated. We calculate the precision for the conditions as the number of correctly generated conditions (correct means all three parts of the condition are correctly generated) divided by the total number of generated conditions. We calculate the recall for conditions as number of correctly generated conditions divided by the number of conditions that should have been generated. The system automatically calculates a precision and recall value for both the return-clause name list and conditions list. For example, if AskOntos had translated the sample query above into an intermediate query with return-clause names Price, Mileage, Make, Model, and Year, (notice that Color is missing and that Model has been incorrectly added) the return-clause names for this query would have a precision of 80%, and a recall of 80%. We calculate precision and recall for each domain as well as for all queries. We also calculate combined precision and recall values (not distinguishing between the two query parts) and the percentage of queries that were translated with 100% accuracy, with partial accuracy, and with 0% accuracy.

4.3 Results

Figure 4.1 shows the translation accuracy from the third round of test queries. The combined precision is 88% and the combined recall is 81%. For the return-clause names, the total precision is 90% and the recall is 90%. For the conditions, the total precision is 86% and the recall is 71%. AskOntos translated 64% of the test queries with 100% accuracy, 32% with partial accuracy, and 4% with 0% accuracy.

By way of comparison, the authors of NaLIX [21], which also converts a natural-language query into an XQuery expression, report an average precision of 83% and an average recall of 90%. While the results from NaLIX look comparable to those of AskOntos,

	Precision	Recall	
Cars	0.89	0.83	Return-Clause Names Conditions
	0.75	0.63	
Houses	0.92	1.00	
	0.90	0.69	
Countries	0.96	1.00	
	0.92	1.00	
Movies	0.75	0.80	
	0.60	0.33	
Diamonds	0.91	0.91	
	1.00	0.85	
All	0.90	0.90	
	0.86	0.71	
All	0.88	0.81	Combined

Queries with 100% accuracy: 64%
 Queries with partial accuracy: 32%
 Queries with 0% accuracy: 4%

Figure 4.1: Experimental results for AskOntos.

their evaluation procedures differ considerably, making a direct comparison difficult. First, since the authors of NaLIX focus their experiments on how well, on average, their system returns correct query results, they report an average precision and recall. Because we focus on how well AskOntos correctly translates the return-clause names and the conditions of a query, we feel it more appropriate to report a precision and recall from the collective number of return clause names and conditions—following the standard of SENSEVAL-3’s Logic Forms task. Second, the testing procedures used by the authors of NaLIX consisted of subjects completing two blocks of search tasks, each block having nine tasks. For each task, subjects had five minutes to read the task, formulate a query, analyze the results, evaluate the results, and then were free to continually reformulate the query to improve precision and recall if they desired. The authors of NaLIX report that each subject was able to formulate a natural-language query acceptable by NaLIX on the first attempt for about half of the search tasks. In our experiments, queries were submitted either on paper or by email (whichever the subject preferred). Queries were not modified before calculating precision and recall values, and, for our experiments, the subjects received no system feedback.

4.4 Issues

Training and experimental queries revealed several underlying difficulties and limitations of AskOntos. The subsections below give examples that illustrate these problems. The problems are organized according to whether they are system issues, domain-ontology issues, or English-language issues.

4.4.1 System Issues

Some limitations and difficulties of AskOntos are caused from system techniques and algorithms, including the use of lexicons, regular expressions, and the extraction heuristics.

Lexicons allow system developers to conveniently list a wide variety of values that appear in a given domain. Although they can be quite extensive, it is nearly impossible to create a lexicon that covers everything a user might ask about in a query. One user, for example, asked about houses that have trees. Since “trees” was not in the *Feature* lexicon

(and perhaps rightly so), the question was not translated correctly. Another difficulty with lexicons is maintenance. With over 3,000 entries in the lexicon for movie titles, AskOntos does a decent job of recognizing movie titles for the Movie ontology, but needs to be updated constantly to recognize new movies. Another problem is that because lexicons may have such a wide range of different phrases, it is difficult to canonicalize values recognized by lexicons. This creates a problem, for example, if a user searches for a car with “chrome wheels” and a car-ad record specifies that it has “chrome rims”. Both strings are recognized by the car accessory lexicon, but without canonicalizing them, so that synonymous terms become identical, they do not match in the search.

The Ontos heuristics can be problematic. Ontos relies on context, keyword, and operation matches to disambiguate recognized values, but does poorly when queries do not provide any of these clues. For example, consider the query “Are there any Ford mustangs, 2002, that are red?” The Car Ad ontology matches the term “2002” as a possible *Year*, *Price*, and *Mileage* value. According to the heuristics, since there are no contextual clues in the text, AskOntos chooses arbitrarily, and it happens to choose *Price*. One solution to this problem would be to embed domain-specific heuristic information into Ontos. The Car Ad ontology could have a heuristic that says if the match starts with “19” or “20” and is four digits long, then, in the absence of contradictory evidence such as a leading dollar sign, it is a *Year* value. Unfortunately, having Ontos handle each ontology with domain-specific heuristics is not scalable.

Another issue related to heuristics can be thought of as a heuristic tug of war; some heuristics work well in some cases, but not so well in others. As an example, consider the heuristic that decides whether a recognized aggregation operation phrase is valid. Suppose an aggregation operation phrase is only accepted if there is a recognized value or keyword by the same data frame. This rule, however, fails to find the $\min(\textit{Price})$ constraint indicated in the query “How much is the cheapest Ford that’s newer than 1995?” The word “cheapest” matches the *Price* minimum aggregation operation expression, but there are no recognized *Price* values or keywords. On the other hand, suppose there need not be a recognized value or keyword for an aggregation operation phrase to be accepted. This makes queries such as “I want a 2-story house $> 2,000$ sq ft, and it should at least have a fireplace”

have a false-positive aggregation operation match because “least”, in this example, matches the *Stories* minimum aggregation operation expression.

Another problem with heuristics that arises occasionally is the unintentional acceptance of a keyword or value, which causes an object-set name or value to be included in the query in error. For example, when processing the query “I want a diamond with a carat > 1.2 and a price no higher than \$4,000,” Ontos recognizes “I” as a diamond color (“I” is a valid diamond color) and erroneously includes *Color* in the query, both as a condition (Color, =, I) and as an object set name in the return-clause.

4.4.2 Domain Issues

Some limitations and difficulties of AskOntos come from the use of ontologies. These difficulties include deciding how many concepts a domain should encompass, deciding what should be included in each concept, and the problem of inconsistencies between the user’s mental ontology and the implemented extraction ontology.

Because subjects in the empirical study were given sample web pages on which to base their queries, it was rare that a concept was referenced that was not covered by an extraction ontology. One example of a query that did mention a concept not covered was, “Who played the voice of Aslan?” Neither the concept of a person’s voice nor the concept of a character in a movie were covered in the Movie Ontology. This, however, was intentional since that information is not typically found on web pages with records of movie data.

Another difficulty with ontologies is in deciding what should and should not be considered part of a concept. For example, should England be identified by a country ontology as a country even though it is not technically a country? It is an administrative division, or constituent country of the UK. This problem is related to the problem of discrepancies between the user’s mental model and the actual extraction ontology. If the user asks for a diamond with a “high certification,” for example, AskOntos would not handle the query correctly because diamond certifications are not necessarily associated with the concepts of high or low. Some companies have a better reputation for certifying diamonds, but whether

one company is better than the other is subjective. The user's mental ontology is different from the implemented ontology.

4.4.3 English Issues

Humans can ignore (or ask for help to resolve) ambiguous phrases. Any system that tries to translate English into something machine-computable, however, will struggle with ambiguities. Subjects in our experiments provided several examples. Consider this query about diamonds: “not more than \$2,000, very high clarity, at least 1.5 carat.” How should AskOntos decide what “high clarity” is (clarity is measured by a scale ranging from F1 to I3, with many gradations in between). Another tricky query is, “Where is Peru?” Acceptable answers could include the name of its continent, the name of its hemisphere, the names of its neighboring countries, or other location indicators. The question is not specific enough. Similarly, consider this query about movies, “more stars than 2 lasting more than 90 minutes.” Did the user mean more than two actors or more than two critic rating stars? It is a very challenging problem indeed to have AskOntos correctly translate queries like these.

Chapter 5

CONCLUSION AND FUTURE WORK

We have created a system called AskOntos that provides a natural-language, free-form, query interface to extracted values from semantically annotated web pages. AskOntos uses a novel approach to query processing in that it applies information extraction to queries by way of extraction ontologies. The use of extraction ontologies facilitates query translation by allowing queries to be matched to an appropriate domain context. Using the extracted values from the appropriate extraction ontology, AskOntos can formulate an intermediate query. The intermediate query specifies return-clause names, conditions, and aggregation functions. With these fundamental query parts, AskOntos can generate an XQuery expression tailored for execution over previously extracted web page data. Querying over extracted values allows AskOntos to return record values of interest rather than entire documents.

Experimental results show that AskOntos translated queries with a combined precision of 88% and a recall of 81%. Return-clause names were translated correctly with a precision of 90% and a recall of 89%. Conditions were translated with a precision of 85% and a recall of 73%. AskOntos translated 64% of the test queries with 100% accuracy, 32% with partial accuracy, and 4% with 0% accuracy.

5.1 Future Work

As this is the first implementation of AskOntos, there are many areas where the system could be improved. We list some of the possibilities as our future work.

Feedback. A feedback mechanism could help users reformulate mistranslated queries. Feedback could consist of simply highlighting the extracted parts of the user query

as well as returning the data that results from running the user query as it is translated. Highlighting recognized parts of the user query could quickly instruct the user about the system's abilities.

Disjunctions and Negations. AskOntos could be improved by enabling it to correctly process queries with disjunctions and negations. One simple yet effective way to do this would be to provide an interface where users could specify lists of disjunctive terms and terms to negate, similar to Google's interface [12].

Fuzzy Queries. Along the lines of improving the system's ability to handle fuzzy or ambiguous queries, one capability that can be added is to appropriately handle queries such as "Find me a Toyota Camry that's older than 2000 and costs around \$6,000." The fuzzy part is knowing what the user meant by "around." One solution might be to compute 10% of the value (600 in this case), and create a query using plus or minus 10% of the value as part of the constraints, (Price, >, 5400) and (Price, <, 6600) in this case. This solution, however, has the standard pitfalls that accompany making fuzzy queries crisp. Therefore, an alternative solution might be to adopt research results from fuzzy set theory or approximate query answering.

Units. Currently AskOntos deals with value units by having the system pass recognized right and left context phrase to the canonicalization operations. Each canonicalization operation has the responsibility of checking those two arguments for clues about units for the extracted value and then converting it to canonical units. One problem with this is that a query such as, "I want a car where the number of kilometers on the odometer is 100K," specifies the units, but they are not immediately to the left or right of the value. While this sample query is a bit unusual, it illustrates a limitation in the current implementation of the system. One solution to this problem is to pass all matched keywords and right and left context phrases to the canonicalization operations, giving the operations more information from which to derive the units.

Spell Checker. While AskOntos does not require any grammatical correctness of user queries, it does require correct spelling. The robustness of the system could be improved by applying software that suggests correct spellings of misspelled words.

Bibliography

- [1] Z. Bar-Yossef, Y. Kanza, Y. Kogan, W. Nutt, and Y. Sagiv. Querying semantically tagged documents on the world wide web. In *Proceedings of the 4th Workshop on Next Generation Information Technologies and Systems (NGITS99)*, pages 2–19, Zikhron-Yaakov, Israel, July 1999.
- [2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific America*, 284(5):34–43, 2001.
- [3] A. Bernstein, E. Kaufmann, A. Gohring, and C. Kiefer. Query ontologies: A controlled English interface for end-users. In *Proceedings of the 4th International Semantic Web Conference*, pages 112–126, Galway, Ireland, November 2005.
- [4] S. Conlon, J. Conlon, and T. James. The economics of natural language interfaces: Natural language processing technology as a scarce resource. *Decision Support Systems*, 38(1):141–159, October 2004.
- [5] A. Copestake and K.S. Jones. Natural language interfaces to databases. *Knowledge Engineering Review*, 5(4):225–249, 1990.
- [6] D. Embley. Programming with data frames for everyday data items. In *Proceedings of the 1980 National Computer Conference*, pages 301–305, Anaheim, California, May 1980.
- [7] D. Embley, E. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y. Ng, and R. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering*, 31(3):227–251, 1999.

- [8] D. Embley, Y. Ng, and L. Xu. Recognizing ontology-applicable multiple-record web documents. In *Proceedings of the 20th International Conference on Conceptual Modeling*, pages 27–30, Yokohama, Japan, November 2001.
- [9] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
- [10] N.E. Fuchs and R. Schwitter. Attempto controlled English (ACE). In *Proceedings of the First International Workshop on Controlled Language Applications*, pages 124–136, Katholieke Universiteit Leuven, Belgium, March 1996.
- [11] D. Goldschmidt and M. Krishnamoorthy. Architecting a search engine for the semantic web. In *Workshops of the Twentieth National Conference on Artificial Intelligence, sponsored by AAAI*, pages 116–119, Pittsburgh, Pennsylvania, July 2005.
- [12] <http://google.com>.
- [13] T.R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1999.
- [14] R. Guha, R. McCool, and E. Miller. Semantic search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 700–709, Budapest, Hungary, May 2003.
- [15] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Proceedings of the 14th International Conference on World Wide Web*, pages 902–903, Chiba, Japan, May 2005.
- [16] A. Gupta and R. Jain. Visual information retrieval. *Communications of the ACM*, 40(5):70–79, May 1997.
- [17] J. Heflin and J. Hendler. Searching the web with SHOE. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop*, pages 35–40, Menlo Park, California, 2000.

- [18] H. Kamp and U. Reyle. *Disourse to Logic: Introduction to Modeltheortic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer Academic Publishers, Dordrecht, 1993.
- [19] M. Klein and A. Bernstein. Towards high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36, January 2004.
- [20] Y. Li, H. Yang, and H. Jagadish. Schema-free XQuery. In *Proceedings of the THirteenth International Conference on Very Large Data Bases*, pages 72–83, Toronto, Canada, August 2004.
- [21] Y. Li, H. Yang, and H. Jagadish. Constructing a generic natural language interface for an XML database. In *Proceedings of International Conference on Extending Database Technology*, Munich, Germany, March 2006. (in press).
- [22] D. Lin. Dependency-based evaluation of MINIPAR. In *Proceedings of the Language Resources and Evalutation Conference Workshop on the Evaluation of Parsing Systems*, pages 48–56, Granada, Spain, May 1998.
- [23] Web ontology language (OWL). <http://www.w3.org/TR/owl-features/>.
- [24] V. Rus. A first evalutaion of logic form identification systems. In *Proceedings of the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 37–40, Barcelona, Spain, July 2004.
- [25] <http://www.senseval.org/>.
- [26] M. Vargas-Vera and E. Motta. AQUA ontology-based question answering system. In *Proceedings of the Third International Mexican Conference on Artificial Intelligence*, pages 26–30, Mexico City, Mexico, April 2004.
- [27] T. Walker and D. Embley. Automatic location and separation of records: A case study in the genealogical domain. In *Proceedings of the International Workshop on Conceptual Model-directed Web Information Integration and Mining*, pages 302–313, Shanghai, China, November 2004.

- [28] A. Wessman, S. Liddle, and D. Embley. A generalized framework for an ontology-based data-extraction system. In *Proceedings of the Fourth International Conference on Information Systems Technology and its Applications*, pages 239–253, Palmerston North, New Zealand, May 2005.
- [29] <http://www.w3.org/TR/xquery/>.

Appendix A

Instruction Packet

This appendix includes the instruction packet given to subjects who participated in the experiments. The first page provides instructions for writing the query, the second page provides space for the queries to be written, and the remaining pages give sample web pages for each of the five domains.

AskOntos Experimental Query Solicitation

We are collecting queries to test AskOntos, a query processing system. Attached are sample web pages for five domains (car ads, house ads, countries, movies, and diamond ads) from which the query results may come.

Expectations:

1. The queries should be database queries (i.e. “Find me...”, “List...” etc.).
2. The results can only come from data similar to the data in the attached web pages.
3. Queries must be in English. They may be incomplete sentences and need not be grammatically correct. They may also include common symbol such as <, >, and \$.

Limitations:

1. No metadata queries, such as “List the characteristics of diamonds.”
2. No queries requiring knowledge outside the given information, such as “Find me a diamond my fiancé will like.”
3. No negations (examples: “... everything but...”, “... not ...”).
4. No disjunctions (examples: “...or...”, “...one of...”).

Give five queries for each domain, 25 queries altogether. You may write your queries on the paper provided or in an email message to: *markvickers@gmail.com*.

Question Form

Domain 1: Car Ads (typical car ad information)

- 1.
- 2.
- 3.
- 4.
- 5.

Domain 2: Houses Ads (typical house ad information)

- 1.
- 2.
- 3.
- 4.
- 5.

Domain 3: Countries (typical almanac information)

- 1.
- 2.
- 3.
- 4.
- 5.

Domain 4: Movies (not theatre info, just typical things associated with movies themselves)

- 1.
- 2.
- 3.
- 4.
- 5.

Domain 5: Diamonds (remember the four C's, cut, clarity, color, and carat)


- 1.
- 2.
- 3.
- 4.
- 5.

CAR AD DOMAIN

April 10
397 vehicles
132 jobs
176 homes

OnlineAthens

ATHENS BANNER-HERALD



BONA FIDE CLASSIFIED

Athens, GA

[NEWS](#) | [SPORTS](#) | [DOGBYTES](#) | [ROCKATHENS](#) | [CLASSIFIEDS](#) | [JOBS](#) | [HOMES](#) | [AUTOS](#) | [CITY GUIDE](#)

[Classifieds](#)
[Real Estate Sales](#)
[Real Estate For Rent](#)
[Employment](#)
[Financial](#)
[Transportation](#)
[Rec. Vehicles](#)
[Merchandise](#)
[Garage/Yard Sales](#)
[Agricultural](#)
[Pets & Livestock](#)
[Personals](#)
[Announcements](#)
[Legal Notices](#)
[Service Directory](#)
[Marketplace](#)
[Homes](#)
[Jobs](#)

TRANSPORTATION

(385 results) - Displaying 1-25

[Previous Page](#)
[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)
[Next Page](#)

[Clear My List](#)
[View My List](#)

Price	Year	Make & Model	Description
\$2,000	1984	Dodge	DODGE W100 1984. 4x4 Pickup. 6" lift 12.5x35" mud tires. Runs good. Good hunting truck. \$2,000 cash. 706-769-4466. Add to My List
\$19,800	2002	TOYOTA TUNDRA	TOYOTA 4WD V8 2002, SR5 Tundra, regular cab. 8' bed. Loaded with upgrades. 100k warranty. Line-X. \$19,800. 706-769-4323. Add to My List
\$2,550	1982	CHEVROLET BLAZER	CHEVROLET BLAZER SILVERADO K5 1982. 4x4. 4 speed. Full size. Black. Cold AC. 350 V8. Tow package w/ brakes. Tape. Looks & runs great. Only 155K mi. \$2,550. 706-372-6579 or 706-540-0939. Add to My List

'88 TOYOTA
Landcruiser, Automatic transmission/doors/windows. 2 owner car, good condition. \$5400 OBO. 801-561-2118.

'90 VOLKSWAGON
Golf GO Hatchback. New tires, needs minor cosmetic work, \$1500 OBO. 364-6485 or 550-7914.

'92 HONDA
Accord, 4 dr, 5 spd, runs great, needs minor work, \$1200 obo. Call 502-6519

'92 SUBARU
White Legacy Wagon, runs good, minor rust, needs minor work. \$1000 OBO. 801-949-0008

'93 TOYOTA
Pick-up, 2 wheel drive, 5 speed, a/c bed liner, new tires, great transportation! \$128k miles. \$2800. 487-8730.

'95 BMW MOTORCYCLE
R-80RT, 20K miles, excell. cond., \$4299 obo. Call 230-6487

1-15 from 23 results.
[next 15 >](#)


HOUSE AD DOMAIN

3 homes viewed this visit.
[Save for your next visit](#)

[Back to results](#)
[Save this home for future visits](#)
[Send to a friend](#)

[Home details](#)
[Photos and tours](#)
[Floor plans](#)
[Driving directions](#)


Chaumont SR by Richmond American Homes
From **\$289,990**

3 br 2.5 ba 2 gr



From 2,007 sq. ft.
2-story home
Ready to build

[Free brochure](#)


Shadow Ridge at Traverse Mountain
Saratoga Springs, UT 84043
[View Community details](#)



Exterior view



Floor plan



Richmond American Homes
2682 West Shady Hollow Lane
Saratoga Springs, UT 84043
[Visit builder's website](#)

Gorgeous French country two-story home with 9 ft ceilings on main floor. Open kitchen overlooking dining area close to family room. Gorgeous master suite with walk-in closets.

[Request more info](#)

Features
Master Bed Upstairs

Category: House for sale (with acreage)
Year: 2005
Street: 27 Regal Regency Ct.
City: Rising Sun, Maryland, MD

Contact: Tom Armour
Address: 27 Regal Regency Court
Location: Rising Sun, Maryland, MD 57946
Phone #1: 410-287-5500

Bedrooms: 4
Bathrooms: 3.5
Stories: 2
Acres: 2.166
Sq. Feet: 4,000
Price: \$ 649,000. USD

Office, gourmet kitchen, media center central air, central vac, stone fireplace, french doors, tile, hardwood floors, carpet, jacuzzi tub, cathedral and vaulted ceilings and a three car garage on a wooded lot.



HousesForSale.com is the logical website to find "Houses For Sale"

[Home](#) [About Us](#) [Contact Us](#) [Terms of Use](#) [Privacy Policy](#) [News](#) [Links](#) [Agents](#)

Copyright © 2000-2005 LinkedAds.com, LLC. All Rights Reserved.

All logos, trademarks, banners, and photos are property of their respective owners.

Although an effort is made to ensure accuracy, we are not responsible for errors, omissions, or typographical errors.



COUNTRY DOMAIN

Nations

[Nations](#) . [explore](#) . [home](#)

Nation	Capital	Continent	Area	Population	Currency	Language
Afghanistan	Kabul	Asia	250,000 sq. mi. (647,500 sq. km.)	29,547,078	afghanis	Afghan Persian (Dari), Pashtu
Albania	Tiranë	Europe	11,100 sq. mi. (28,750 sq. km.)	3,544,808	leks	Albanian, Greek
Algeria	Algiers	Africa	919,600 sq. mi. (2,381,740 sq. km.)	32,357,089	dinars	Arabic, French, Berber Dialects
Andorra	Andorra la Vella	Europe	174 sq. mi. (450 sq. km.)	69,865	euros	Catalan, French, Castilian
Angola	Luanda	Africa	481,400 sq. mi. (1,246,700 sq. km.)	10,978,552	kwanzas	Portuguese, African dialects
Argentina	Buenos Aires	South America	1,068,300 sq. mi. (2,766,890 sq. km.)	39,144,753	peso	Spanish, English, Italian
Armenia	Yerevan	Asia	11,500 sq. mi. (29,800 sq. km.)	3,325,307	drams	Armenian
Australia	Canberra	Australia	2,967,910 sq. mi. (7,686,850 sq. km.)	19,913,144	Australian dollars	English, aboriginal languages
Austria	Vienna	Europe	32,380 sq. mi. (83,860 sq. km.)	8,174,762	euros	German
Azerbaijan	Baku	Asia	33,440 sq. mi. (86,600 sq. km.)	7,868,385	manats	Azeri, Russian, Armenian



NATIONAL GEOGRAPHIC | E381

[Order Custom Maps by Mail >>](#)

[MAPMACHINE HOME](#)
[SEARCH AND BROWSE](#)
[VIEW AND CUSTOMIZE](#)
[MY SAVED MAPS](#)
[COUNTRY PROFILES](#)

[MapMachine Home](#) > [Country Profiles](#) > [Afghanistan](#)

Country Profiles

Afghanistan

Islamic Republic of Afghanistan

[View Dynamic Map](#) | [View Atlas Plate](#)
[Printable Outline Map](#) | [CIA World Factbook Entry](#)

Since Alexander the Great, invading armies and peaceful migrations have brought in diverse peoples to this Central Asian crossroads. As a result, Afghanistan is a country of ethnic minorities: Pashtun (58 percent), Tajik (25 percent), Hazara (10 percent), and Uzbek (5 percent). The towering Hindu Kush range dominates and divides Afghanistan. The northern plains and valleys are home to Tajiks and Uzbeks. Pashtuns inhabit the desert-dominated southern plateaus. Hazaras live in the central highlands. Kabul, south of the Hindu Kush, is linked by narrow passes to the northern plains.



In 1989 the nine-year Soviet occupation ended, and Muslim rebels toppled the communist regime in 1992, after which rival groups vied for power. From among the various factions arose the Taliban ("students of religion"), a militant Islamic movement. The Taliban seized Kabul in 1996 and imposed Islamic punishments, including amputation and stoning, and banned women from working. In 2001 the Taliban destroyed giant Buddha statues at Bamian in defiance of international efforts to save them. Three weeks after the September 11 attacks on New York and Washington, D.C., the U.S. and Britain bombed terrorist camps

Area
652,090 sq km (251,773 sq mi)

Population
28,717,000

Population Density
47 per sq km

Capital
Kabul 2,956,000

Religion
Sunni and Shiite Muslim














Languages
Pashtu, Afghan Persian (Dari), Uzbek, Turkmen, 30 minor languages

Currency
afghani

MOVIE DOMAIN

USA

Major Films

	Rating	Year	Title		Director
	★★★★★	1999	Toy Story 2		John Lasseter
	★★★★★	1999	Being John Malkovich		Spike Jonze
	★★★★★	1998	Rushmore		Wes Anderson
	★★★★★	1998	Saving Private Ryan		Steven Spielberg
	★★★★★	1993	Visions of Light: The Art of Cinematography		Stuart Samuels
	★★★★★	1997	Titanic		James Cameron
	★★★★★	1997	Boogie Nights		Paul Thomas Anderson



25. The Motorcycle Diaries (2004)



R **2 Hr 8 Mn**
 USA / English
Drama
 By: **Walter Salles**

Gael Garcia Bernal Rodrigo de la Serna
 Mia Maestro Mercedes Moran

[Trade](#) Add to my [Collection](#) | [Wishlist](#)

26. Lost in Translation (2003)



R **1 Hr 45 Mn**
 USA / English
Drama, Romance
 By: **Sofia Coppola**

Scarlett Johansson Bill Murray
 Anna Faris Giovanni Ribisi

[Trade](#) Add to my [Collection](#) | [Wishlist](#)

27. Kingdom of Heaven (2005)



R **2 Hr 25 Mn**
 USA / English
Drama, Action
 By: **Ridley Scott**

Orlando Bloom Eva Green
 Liam Neeson Brendan Gleeson

[Trade](#) Add to my [Collection](#) | [Wishlist](#)

28. Batman Begins (2005)



PG13 **2 Hr 14 Mn**
 USA / English
Action
 By: **Christopher Nolan**

Christian Bale Ken Watanabe
 Cillian Murphy Morgan Freeman

[Trade](#) Add to my [Collection](#) | [Wishlist](#)

(
Liber
Conserve

Story D
Fun & T

Ha
Stror
Fii
Delicat



DIAMOND DOMAIN



diamond results

Each of these diamonds comes with an EGL or GIA Certificate unless otherwise specified. **All diamonds come from our own inventory, not a shared database.** For questions please call us TOLL FREE at 877-579-BEST or email us at info@bestgem.com. For detailed information, click on the diamond of your choice.

Home

Loose Diamonds

- [Diamond Search](#)
- [Color Diamonds](#)
- [The 4 C's](#)
- [Expert Advice](#)

Rings

Pendants

Earrings

Sort results by... ▼



SKU: PR592524
Weight: 1.07ct
Color: F
Clarity: SI1
Cert: EGL
on sale
was \$5,130.00
Now \$4,617.00
[View Diamond](#)



SKU: PR566538
Weight: 1.08ct
Color: F
Clarity: SI2
Cert: EGL
on sale
was \$3,900.00
Now \$3,510.00
[View Diamond](#)



SKU: PR1010
Weight: 2.01ct
Color: G
Clarity: VS2
Cert: EGL
on sale
was \$0.00
Now \$14,000.00
[View Diamond](#)

QUICK SEARCH

Loose Diamonds ▼

Round Cut Diamonds ▼

BUDGET Any Price ▼

SIZE Any Size ▼

diamond

For Phone Orders
Please Mention Code
25591

[contact feedback](#)

[diamond shopping](#)

[diamond safe](#)

FOR ASSISTANCE AND
PHONE ORDERS CALL
1-800-437-8444

Click a Diamond to View Actual Diamond and Certificate
57 diamonds found

0.82 ct On Sale

Good Cut

EGL certified

[dsm9131](#)

SI1 Clarity
D Color

\$2,956

\$2,809

0.84 ct On Sale

Tolkowsky Ideal Cut

EGL certified



SI2 Clarity
D Color

\$2,938

\$2,791

0.70 ct On Sale

Good Cut

EGL certified



VS2 Clarity
D Color

\$2,824

\$2,683

0.70 ct On Sale

Good Cut

EGL certified



VS2 Clarity
D Color

\$2,824

\$2,683

0.70 ct On Sale

Good Cut

EGL certified



VS2 Clarity
D Color

\$2,824

\$2,683

0.73 ct On Sale

Good Cut

EGL certified



VS1 Clarity
F Color

\$2,783

\$2,644

Appendix B

Experimental Results for the Penultimate Version of AskOntos

Round 2			Round 3		
	Precision	Recall		Precision	Recall
Cars	0.94	0.92	Cars	0.88	0.79
	0.88	0.63		0.73	0.58
Houses	0.94	0.91	Houses	0.10	1.00
	0.93	0.82		1.00	0.77
Countries	0.86	0.79	Countries	0.95	0.86
	0.79	0.79		0.92	1.00
Movies	0.78	0.97	Movies	0.76	0.87
	0.83	0.76		0.67	0.44
Diamonds	0.89	1.00	Diamonds	0.91	0.91
	0.92	0.88		1.00	0.85
All	0.88	0.91	All	0.89	0.86
	0.88	0.77		0.87	0.72
All	0.86	0.82	All	0.88	0.80
Combined			Combined		
Queries with 100% accuracy: 34%			Queries with 100% accuracy: 60%		
Queries with partial accuracy: 62%			Queries with partial accuracy: 36%		
Queries with 0% accuracy: 4%			Queries with 0% accuracy: 4%		

Figure B.1: Experimental results from both the second and third round for the penultimate version of AskOntos.

Appendix C

All Third-Round Queries with Hand-Generated and System-Generated Translations for AskOntos version 2

Round 3 Queries, System Version 2

List all cars with price < \$8000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Model, Price, Make	Model, Price, Make
Conditions	1	1	(Price < 8000)	(Price < 8000)

List all cars where the make is Toyota

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Model, Make	Model, Make
Conditions	1	1	(Make = toyota)	(Make = toyota)

List all cars where the model is Camry

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Model, Make	Model, Make
Conditions	1	1	(Model = camry)	(Model = camry)

List all cars where the color is blue

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Model, Color, Make	Model, Color, Make
Conditions	1	1	(Color = blue)	(Color = blue)

List all cars with year > 2004

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Year, Model, Make	Year, Model, Make
Conditions	1	1	(Year > 2004)	(Year > 2004)

List all houses with price < \$200001

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price	Price
Conditions	1	1	(Price < 200001)	(Price < 200001)

List all houses with rooms > 4

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	NumBR	NumBR
Conditions	1	1	(NumBR > 4)	(NumBR > 4)

List all houses with bathrooms > 2

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	NumBA	NumBA
Conditions	1	1	(NumBA > 2)	(NumBA > 2)

List all houses in Utah

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	State	State
Conditions	1	1	(State = utah)	(State = utah)

List all houses with square footage > 2000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	SquareFeet	SquareFeet
Conditions	1	1	(SquareFeet > 2000)	(SquareFeet > 2000)

List the country with the capital Washington DC

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Capital, Name	Capital, Name
Conditions	1	1	(Capital = washington dc)	(Capital = washington dc)

List all countries where the continent is Asia

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Continent, Name	Continent, Name
Conditions	1	1	(Continent = asia)	(Continent = asia)

List all countries with population > 90000000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Population, Name	Population, Name
Conditions	1	1	(Population > 9)	(Population > 9)

List all countries where English is the language

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.67	1	Name, Language	Continent, Name, Language
Conditions	1	1	(Language = english)	(Language = english)

List all countries with euros as the currency

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Name, Currency	Name, Currency
Conditions	1	1	(Currency = euro)	(Currency = euro)

List all movies with the year > 2000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Year, Title	Year, Title
Conditions	1	1	(Year > 2000)	(Year > 2000)

List all movies with rating > 3

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.67	1	CriticRating, Title	CriticRating, Title, Rating
Conditions	0	0	(CriticRating > 3)	

List all movies where the director is Spike Jonze

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Title, Director	Title, Director
Conditions	1	1	(Director = spike jonze)	(Director = spike jonze)

List all movies with the word TOY in the title

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Title	Title
Conditions	0	0	(Title = like_Toy)	

List all movies with the category action

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Genre, Title	Genre, Title
Conditions	1	1	(Genre = action)	(Genre = action)

List all diamonds with color of F

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Color	Color
Conditions	1	1	(Color = f)	(Color = f)

List all diamonds < \$500

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price	Price
Conditions	1	1	(Price < 500)	(Price < 500)

List all diamonds with clarity of VS2

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Clarity	Clarity
Conditions	1	1	(Clarity = vs2)	(Clarity = vs2)

List all diamonds > 2ct

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0	0	Carat	
Conditions	0	0	(Carat > 2)	

List all diamonds with color of D

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Color	Color
Conditions	1	1	(Color = d)	(Color = d)

I want a chevy truck costing less than \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	BodyType, Price, Make	BodyType, Price, Make
Conditions	1	1	(Make = chevy), (BodyType = truck), (Price < 5000)	(Price < 5000), (BodyType = truck), (Make = chevy)

List all hondas and toyotas made between 1998 and 2003 that are green

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.67	0.67	Year, Color, Make	Color, Make, Mileage
Conditions	0.5	0.4	(Color = green), (Make = honda), (Year >= 1998), (Year <= 2003), (Make = toyota)	(Make = honda), (Mileage <= 2003), (Mileage >= 1998), (Color = green)

Find all 5 speed manual cars

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.75	1	Model, Feature, Make	Model, Feature, Make, Mileage
Conditions	0.5	0.5	(Feature = manual), (Feature = 5 speed)	(Mileage = 5), (Feature = manual)

List the mini-coopers

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Make	Make
Conditions	1	1	(Make = mini)	(Make = mini)

What are the models from 2005

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.5	0.5	Year, Model	Model, Mileage
Conditions	0	0	(Year = 2005)	(Mileage = 2005)

Show the houses < \$200,000 and > \$100,000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price	Price
Conditions	0	0	(Price < 200000), (Price > 100000)	

Show all houses with more than 4000 sq. ft

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.5	1	SquareFeet	SquareFeet, LotSize
Conditions	0	0	(SquareFeet > 4000)	(LotSize > 4000)

List the houses in Provo

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	City	City
Conditions	1	1	(City = provo)	(City = provo)

Find me houses with master suites and a jacuzzi costing less than \$400K

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price, Features	Price, Features
Conditions	1	0.67	(Price < 400000), (Features = master suites), (Features = jacuzzi)	(Price < 400000), (Features = master suites)

Which houses have 3 stories

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Stories	Stories
Conditions	1	1	(Stories = 3)	(Stories = 3)

What countries speak english?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Name, Language	Name, Language
Conditions	1	1	(Language = english)	(Language = english)

What is the capital and population of Kazakhstan?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Population, Capital, Name	Population, Capital, Name
Conditions	1	1	(Name = Kazakhstan)	(Name = kazakhstan)

What is the biggest country in South America?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Continent, Area, Name	Continent, Area, Name
Conditions	1	1	(Area = max), (Continent = south america)	(Continent = south america), (Area = max)

Which countries use the euro?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Name, Currency	Name, Currency
Conditions	0.5	1	(Currency = euro)	(Currency = euro), (Name = us)

What is the currency of Poland?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Name, Currency	Name, Currency
Conditions	1	1	(Name = Poland)	(Name = poland)

Who played the voice of Aslan?

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0	0	Star, Character	
Conditions	0	0	(Character = Aslan)	

When did Splash premiere

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	0.5	Year, Title	Year
Conditions	0	0	(Title = splash)	

In what movies has Humphrey Bogart played in

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.5	0.5	Star, Title	Title, Director
Conditions	0	0	(Star = humphrey bogart)	(Director = humphrey bogart)

In which movies did Meg Ryan and Tom Hanks play in

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.5	0.5	Star, Title	Title, Director
Conditions	0	0	(Star = meg ryan), (Star = tom hanks)	(Director = meg ryan)

What is the rating of Emperor's new Groove

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.5	0.5	Title, Rating	CriticRating, Rating
Conditions	0	0	(Title = emperor's new groove)	

List all diamonds > 1.5 ct

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Carat	Carat
Conditions	1	1	(Carat > 1)	(Carat > 1)

List all FL and IF diamonds

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Clarity	Clarity
Conditions	1	0.5	(Clarity = fl), (Clarity = if)	(Clarity = fl)

List diamonds between \$300 and \$1000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price	Price
Conditions	1	1	(Price <= 1000), (Price >= 300)	(Price <= 1000), (Price >= 300)

Show the diamonds > \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	1	1	Price	Price
Conditions	1	1	(Price > 5000)	(Price > 5000)

Show princess cut < \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
Return-Clause Names	0.67	1	Price, Shape	Price, Shape, Cut
Conditions	1	1	(Shape = princess), (Price < 5000)	(Price < 5000), (Shape = princess)

Total:

---	PRECISION	RECALL
Return-Clause Names	0.90	0.90
Conditions	0.85	0.71

projection Correct: 79

projection Attempted: 88

projection Shoul have got: 88

selection Correct: 46

selection Attempted: 54

selection Shoul have got: 65

Appendix D

All Third-Round Queries with Hand-Generated and System-Generated Translations for AskOntos version 1

Round 3 Queries, System Version 1

List all cars with price < \$8000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Price, Make	Model, Price, Make
SELECT	1	1	(Price < 8000)	(Price < 8000)

List all cars where the make is Toyota

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Make	Model, Make
SELECT	1	1	(Make = toyota)	(Make = toyota)

List all cars where the model is Camry

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Make	Model, Make
SELECT	1	1	(Model = camry)	(Model = camry)

List all cars where the color is blue

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Color, Make	Model, Color, Make
SELECT	1	1	(Color = blue)	(Color = blue)

List all cars with year > 2004

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Model, Make	Model, Year, Make
SELECT	1	1	(Year > 2004)	(Year > 2004)

List all houses with price < \$200001

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price < 200001)	(Price < 200001)

List all houses with rooms > 4

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	NumBR	NumBR
SELECT	1	1	(NumBR > 4)	(NumBR > 4)

List all houses with bathrooms > 2

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	NumBA	NumBA
SELECT	1	1	(NumBA > 2)	(NumBA > 2)

List all houses in Utah

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	State	State
SELECT	1	1	(State = utah)	(State = utah)

List all houses with square footage > 2000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	SquareFeet	SquareFeet
SELECT	1	1	(SquareFeet > 2000)	(SquareFeet > 2000)

List the country with the capital Washington DC

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Capital, Name	Capital, Name
SELECT	1	1	(Capital = washington dc)	(Capital = washington dc)

List all countries where the continent is Asia

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Continent, Name	Continent, Name
SELECT	1	1	(Continent = asia)	(Continent = asia)

List all countries with population > 90000000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Population, Name	Population
SELECT	1	1	(Population > 9)	(Population > 9)

List all countries where English is the language

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	1	Name, Language	Continent, Name, Language
SELECT	1	1	(Language = english)	(Language = english)

List all countries with euros as the currency

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Name, Currency	Currency
SELECT	1	1	(Currency = euro)	(Currency = euro)

List all movies with the year > 2000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Title	Year, Title
SELECT	1	1	(Year > 2000)	(Year > 2000)

List all movies with rating > 3

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	1	CriticRating, Title	CriticRating, Title, Rating
SELECT	0	0	(CriticRating > 3)	

List all movies where the director is Spike Jonze

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Title, Director	Title, Director
SELECT	1	1	(Director = spike jonze)	(Director = spike jonze)

List all movies with the word TOY in the title

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Title	Title
SELECT	0	0	(Title = like_Toy)	

List all movies with the category action

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Genre, Title	Genre, Title
SELECT	1	1	(Genre = action)	(Genre = action)

List all diamonds with color of F

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Color	Color
SELECT	1	1	(Color = f)	(Color = f)

List all diamonds < \$500

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price < 500)	(Price < 500)

List all diamonds with clarity of VS2

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Clarity	Clarity
SELECT	1	1	(Clarity = vs2)	(Clarity = vs2)

List all diamonds > 2ct

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0	0	Carat	
SELECT	0	0	(Carat > 2)	

List all diamonds with color of D

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Color	Color
SELECT	1	1	(Color = d)	(Color = d)

I want a chevy truck costing less than \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	BodyType, Price, Make	BodyType, Price, Make
SELECT	1	1	(BodyType = truck), (Make = chevy), (Price < 5000)	(Price < 5000), (BodyType = truck), (Make = chevy)

List all hondas and toyotas made between 1998 and 2003 that are green

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	0.67	Year, Color, Make	Color, Make, Mileage
SELECT	0.5	0.4	(Color = green), (Make = toyota), (Year >= 1998), (Make = honda), (Year <= 2003)	(Color = green), (Mileage <= 2003), (Mileage >= 1998), (Make = honda)

Find all 5 speed manual cars

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	0.67	Model, Feature, Make	Model, Make, Mileage
SELECT	0	0	(Feature = manual), (Feature = 5 speed)	(Mileage = 5)

List the mini-coopers

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Make	Make
SELECT	1	1	(Make = mini)	(Make = mini)

What are the models from 2005

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.5	0.5	Year, Model	Model, Mileage
SELECT	0	0	(Year = 2005)	(Mileage = 2005)

Show the houses < \$200,000 and > \$100,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	0	0	(Price < 200000), (Price > 100000)	

Show all houses with more than 4000 sq. ft

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	SquareFeet	SquareFeet
SELECT	1	1	(SquareFeet > 4000)	(SquareFeet > 4000)

List the houses in Provo

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	City	City
SELECT	1	1	(City = provo)	(City = provo)

Find me houses with master suites and a jacuzzi costing less than \$400K

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price, Features	Price, Features
SELECT	1	0.67	(Features = jacuzzi), (Features = master suites), (Price < 400000)	(Price < 400000), (Features = master suites)

Which houses have 3 stories

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Stories	Stories
SELECT	1	1	(Stories = 3)	(Stories = 3)

What countries speak english?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Name, Language	Language
SELECT	1	1	(Language = english)	(Language = english)

What is the capital and population of Kazakhstan?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Population, Capital, Name	Population, Capital, Name
SELECT	1	1	(Name = Kazakhstan)	(Name = kazakhstan)

What is the biggest country in South America?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Continent, Area, Name	Continent, Area, Name
SELECT	1	1	(Continent = south america), (Area = max)	(Area = max), (Continent = south america)

Which countries use the euro?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Name, Currency	Name, Currency
SELECT	0.5	1	(Currency = euro)	(Name = us), (Currency = euro)

What is the currency of Poland?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Name, Currency	Name, Currency
SELECT	1	1	(Name = Poland)	(Name = poland)

Who played the voice of Aslan?

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0	0	Star, Character	
SELECT	0	0	(Character = Aslan)	

When did Splash premiere

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Year, Title	Year
SELECT	0	0	(Title = splash)	

In what movies has Humphrey Bogart played in

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.5	0.5	Star, Title	Title, Director
SELECT	0	0	(Star = humphrey bogart)	(Director = humphrey bogart)

In which movies did Meg Ryan and Tom Hanks play in

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	1	Star, Title	Star, Title, Director
SELECT	0.5	0.5	(Star = meg ryan), (Star = tom hanks)	(Director = meg ryan), (Star = tom hanks)

What is the rating of Emperor's new Groove

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.5	0.5	Title, Rating	CriticRating, Rating
SELECT	0	0	(Title = emperor's new groove)	

List all diamonds > 1.5 ct

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Carat	Carat
SELECT	1	1	(Carat > 1)	(Carat > 1)

List all FL and IF diamonds

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Clarity	Clarity
SELECT	1	0.5	(Clarity = fl), (Clarity = if)	(Clarity = fl)

List diamonds between \$300 and \$1000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price <= 1000), (Price >= 300)	(Price <= 1000), (Price >= 300)

Show the diamonds > \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price > 5000)	(Price > 5000)

Show princess cut < \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	1	Price, Shape	Price, Shape, Cut
SELECT	1	1	(Shape = princess), (Price < 5000)	(Price < 5000), (Shape = princess)

Total:

---	PRECISION	RECALL
PROJECT	0.89	0.86
SELECT	0.87	0.72

projection Correct: 76

projection Attempted: 85

projection Should have been: 88

selection Correct: 47

selection Attempted: 54

selection Should have been: 65

Appendix E

All Second-Round Queries with Hand-Generated and System-Generated Translations for AskOntos version 1

Round 2 Queries, System Version 1

List all cars < \$10,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Price, Make	Model, Price, Make
SELECT	1	1	(Price < 10000)	(Price < 10000)

List cars with 4 doors, make is Honda, and mileage < 100,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.75	Model, Feature, Make, Mileage	Model, Make, Mileage
SELECT	1	0.67	(Feature = 4 doors), (Make = honda), (Mileage < 100000)	(Make = honda), (Mileage < 100000)

List cars > \$6,000, < \$15,000, and year is 2002

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Model, Price, Make	Year, Model, Price, Make
SELECT	1	1	(Year = 2002), (Price < 15000), (Price > 6000)	(Price < 15000), (Price > 6000), (Year = 2002)

List cars with automatic transmission, has 4 doors, and make is honda

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	0.67	Model, Feature, Make	Model, Make, Mileage
SELECT	0.5	0.33	(Feature = 4 doors), (Feature = automatic transmission), (Make = honda)	(Mileage = 4), (Make = honda)

List cars with AC, automatic transmission, and year is 2002

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Model, Feature, Make	Model, Year, Feature, Make
SELECT	1	0.67	(Feature = ac), (Year = 2002), (Feature = automatic transmission)	(Feature = ac), (Year = 2002)

List houses in Provo, Utah

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	State, City	State, City
SELECT	1	1	(City = provo), (State = utah)	(City = provo), (State = utah)

List houses with 2 bedrooms and < \$200,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price, NumBR	Price, NumBR
SELECT	1	1	(Price < 200000), (NumBR = 2)	(Price < 200000), (NumBR = 2)

List houses with > 2,000 sq. feet

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	SquareFeet	SquareFeet
SELECT	1	1	(SquareFeet > 2000)	(SquareFeet > 2000)

List houses < \$250,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price < 250000)	(Price < 250000)

List houses with 3 bedrooms, 2 bathrooms, and is newer than 1990

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, NumBA, NumBR	Year, NumBA, NumBR
SELECT	1	1	(Year > 1990), (NumBA = 2), (NumBR = 3)	(Year > 1990), (NumBR = 3), (NumBA = 2)

List countries with Christian religion

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Religion, Name	Religion
SELECT	1	1	(Religion = christian)	(Religion = christian)

List countries with population > 50,000,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Population, Name	Population
SELECT	1	1	(Population > 5)	(Population > 5)

List countries that use the dollar

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Name, Currency	Name
SELECT	0	0	(Currency = dollar)	(Name = us)

List countries that speak English

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.5	Name, Language	Language
SELECT	1	1	(Language = english)	(Language = english)

List countries on the European continent that speak English

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.67	Continent, Name, Language	Continent, Language
SELECT	1	1	(Continent = europe), (Language = english)	(Language = english), (Continent = europe)

Find movies directed by Tom Cruise

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Title, Director	Title, Director
SELECT	0	0	(Director = tom cruise)	

List movies from the year 2006 with PG-13 rating

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	Year, Title, Rating	CriticRating, Year, Title, Rating
SELECT	0.5	0.5	(Rating = pg-13), (Year = 2006)	(Rating = pg), (Year = 2006)

List movies starring Tom Hanks

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.67	1	Star, Title	Title, Star, Director
SELECT	0	0	(Star = tom hanks)	(Director = tom hanks)

List movies from the action category from year 2005

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Genre, Title	Year, Genre, Title
SELECT	1	1	(Year = 2005), (Genre = action)	(Genre = action), (Year = 2005)

List movies < 2hrs with PG-13 rating from romantic comedy category

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.8	1	Genre, Length, Title, Rating	CriticRating, Genre, Length, Title, Rating
SELECT	0.67	0.67	(Length < 2hrs), (Genre = romantic comedy), (Rating = pg-13)	(Genre = romantic comedy), (Length < 2hrs), (Rating = pg)

Find diamonds < \$2,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price < 2000)	(Price < 2000)

Find diamonds with clarity SI2 and color D

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Color, Clarity	Color, Clarity
SELECT	1	1	(Clarity = si2), (Color = d)	(Color = d), (Clarity = si2)

Find diamonds with clarity SI2 and carat <= 0.7

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Carat, Clarity	Carat, Clarity
SELECT	1	1	(Clarity = si2), (Carat <= 0)	(Carat <= 0), (Clarity = si2)

List Diamonds EGL certified with clarity SI2 <\$3,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price, Clarity, Certification	Price, Clarity, Certification
SELECT	1	1	(Price < 3000), (Certification = egl), (Clarity = si2)	(Certification = egl), (Clarity = si2), (Price < 3000)

List Diamonds >\$1,000 and <\$3,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price	Price
SELECT	1	1	(Price < 3000), (Price > 1000)	(Price < 3000), (Price > 1000)

List all cars that have year > 1996 and make is GM and price < \$3000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Model, Price, Make	Year, Model, Price, Make
SELECT	1	1	(Make = gm), (Price < 3000), (Year > 1996)	(Make = gm), (Price < 3000), (Year > 1996)

Find me a truck that has AC, 4 wheel drive, dual air bags, and costs less than \$5000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	BodyType, Price, Feature	BodyType, Price, Feature, Mileage
SELECT	0.6	0.6	(Feature = ac), (BodyType = truck), (Feature = 4 wheel drive), (Feature = dual air bags), (Price < 5000)	(Feature = air), (Mileage = 4), (Feature = ac), (BodyType = truck), (Price < 5000)

Find a car that is Black, is a v8, has rear wheel drive, is from 1970 or before

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Model, Color, Feature, Make	Year, Model, Color, Feature, Make
SELECT	1	0.75	(Year <= 1970), (Color = black), (Feature = v8), (Feature = rear wheel drive)	(Feature = v8), (Year <= 1970), (Color = black)

Find a truck with a full bed, mp3 player, sun roof, and LED/Laser headlights

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0	0	BodyType, Feature	Title
SELECT	0	0	(Feature = full bed), (Feature = sun roof), (Feature = mp3 player), (BodyType = truck), (Feature = led/laser headlights)	(Title = player)

Find a car that has leather seats, chrome rims, and power locks

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Model, Feature, Make	Model, Feature, Make
SELECT	1	0.33	(Feature = power locks), (Feature = chrome rims), (Feature = leather)	(Feature = leather)

3 Bedrooms, 2 Bath, central air, and < \$140,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.75	Price, NumBA, Features, NumBR	Price, NumBA, NumBR
SELECT	1	0.75	(Features = central air), (NumBA = 2), (Price < 140000), (NumBR = 3)	(NumBR = 3), (Price < 140000), (NumBA = 2)

5 Bedrooms, 3 Bath, study, game room, 2 car garage, and < \$250,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.8	Price, GarageSize, NumBA, Features, NumBR	Price, GarageSize, NumBA, NumBR
SELECT	1	0.67	(Price < 250000), (NumBA = 3), (GarageSize = 2), (Features = study), (Features = game room), (NumBR = 5)	(Price < 250000), (NumBR = 5), (GarageSize = 2), (NumBA = 3)

2 Bed, 1 Bath, < \$70,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Price, NumBA, NumBR	Price, NumBA, NumBR
SELECT	1	1	(Price < 70000), (NumBA = 1), (NumBR = 2)	(NumBR = 2), (NumBA = 1), (Price < 70000)

Pool, 3 Bed, 1.5 Bath or more, 1 Acre, and < \$170,000

			User Query	AskOntos Query
---	PRECISION	RECALL	Price, NumBA, LotSize, NumBR, Features	Price, GarageSize, NumBA, LotSize, Features, NumBR
PROJECT	0.83	1	(NumBR = 3), (LotSize = 1), (Features = pool), (Price < 170000), (NumBA >= 1)	(GarageSize = 1), (NumBR = 3), (Features = Pool), (Price < 170000), (NumBA >= 1)
SELECT	0.8	0.8		

2 Acre, trees, 2000 sq ft or more, 4 Bed, 2 Bath, and < \$300,000

			User Query	AskOntos Query
---	PRECISION	RECALL	Price, SquareFeet, NumBA, LotSize, NumBR, Features	Price, SquareFeet, GarageSize, NumBA, LotSize, NumBR
PROJECT	0.83	0.83	(Price < 300000), (NumBR = 4), (SquareFeet >= 2000), (LotSize = 2), (NumBA = 2), (Features = trees)	(NumBR = 4), (SquareFeet >= 2000), (GarageSize = 2), (NumBA = 2), (Price < 300000)
SELECT	0.8	0.67		

Area < 40,000 sq mi and population > 2 million

			User Query	AskOntos Query
---	PRECISION	RECALL	Population, Area	PopDensity, Population, Area
PROJECT	0.67	1	(Population > 2000000), (Area < 40000)	(Population > 2000000), (PopDensity < 40000)
SELECT	0.5	0.5		

Language is spanish, in south america, population > 20 million

			User Query	AskOntos Query
---	PRECISION	RECALL	Population, Continent, Language	Population, Continent, Language
PROJECT	1	1	(Population > 2), (Language = spanish), (Continent = south america)	(Population > 2), (Language = spanish), (Continent = south america)
SELECT	1	1		

In Asia, Area > 100,000 sq mi, and population < 40 million

			User Query	AskOntos Query
---	PRECISION	RECALL	Population, Continent, Area	PopDensity, Population, Continent, Area
PROJECT	0.75	1	(Population < 4), (Continent = asia), (Area > 100000)	(PopDensity > 100000), (Continent = asia), (Population < 4)
SELECT	0.67	0.67		

In Europe, Currency is euros, and Area > 20,000 sq. mi

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	Continent, Area, Currency	PopDensity, Continent, Area, Currency
SELECT	0.67	0.67	(Area > 20000), (Continent = europe), (Currency = euro)	(Currency = euro), (Continent = europe), (PopDensity > 20000)

Language is English, population < 100,000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Population, Language	Population, Language
SELECT	1	1	(Language = english), (Population < 100000)	(Population < 100000), (Language = english)

Rated PG, Made after 2003, < 2 hr, Made in the USA

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	0.75	Year, Length, Country, Rating	Year, Length, Rating
SELECT	1	0.75	(Year > 2003), (Rating = pg), (Length < 2 hr), (Country = USA)	(Rating = pg), (Year > 2003), (Length < 2 hr)

Directed by Steven Spielberg, before 2000, and Rating was >= 4 stars

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.6	1	CriticRating, Year, Director	CriticRating, Year, Star, Rating, Director
SELECT	1	1	(Year < 2000), (Director = steven spielberg), (CriticRating >= 4)	(Director = steven spielberg), (CriticRating >= 4), (Year < 2000)

Action, Sci-Fi, Rated PG-13 or lower, Made after 2000

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Year, Genre, Rating	Year, Genre, Rating
SELECT	1	1	(Year > 2000), (Genre = action, sci-fi), (Rating <= pg-13)	(Rating <= pg-13), (Year > 2000), (Genre = action, sci-fi)

Stars Robin Williams, Rated PG-13 or lower, Rating was ≥ 3 stars

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	CriticRating, Star, Rating	CriticRating, Star, Rating, Director
SELECT	0.67	0.67	(Rating \leq pg-13), (CriticRating \geq 3), (Star = robin williams)	(CriticRating \geq 3), (Rating \leq pg-13), (Director = robin williams)

Longer than 2 hr, Romance, Rating was 5 stars

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.6	1	CriticRating, Genre, Length	CriticRating, Genre, Length, Star, Rating
SELECT	1	1	(CriticRating = 5), (Length > 2 hr), (Genre = romance)	(Length > 2 hr), (Genre = romance), (CriticRating = 5)

0.5 carat, Clarity SI1 or better, princess cut

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	Carat, Clarity, Shape	Carat, Clarity, Shape, Cut
SELECT	1	1	(Carat = 0), (Clarity \geq si1), (Shape = princess)	(Clarity \geq si1), (Carat = 0), (Shape = princess)

Premium Cut, Clarity IF, Color F

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Color, Clarity, Cut	Color, Clarity, Cut
SELECT	1	0.67	(Clarity = if), (Color = F), (Cut = premium)	(Color = f), (Clarity = if)

1.5 carat, Clarity VVS2, Color G or better

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.75	1	Color, Carat, Clarity	Color, Carat, Clarity, Cut
SELECT	0.67	0.67	(Clarity = vvs2), (Color \geq g), (Carat = 1)	(Carat = 1), (Cut \geq good), (Clarity = vvs2)

Cut Tolkowsky Ideal, Clarity VS1 or better, round cut

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	1	1	Clarity, Shape, Cut	Clarity, Shape, Cut
SELECT	0.67	0.67	(Clarity >= vs1), (Cut = tolkowsky ideal), (Shape = round)	(Cut = ideal), (Shape = round), (Clarity >= vs1)

< \$2000, Clarity VS2 or better, carat 1.0, round cut, Color I or better

---	PRECISION	RECALL	User Query	AskOntos Query
PROJECT	0.83	1	Color, Carat, Price, Clarity, Shape	Price, Carat, Color, Clarity, Shape, Cut
SELECT	1	1	(Color >= i), (Shape = round), (Price < 2000), (Carat = 1), (Clarity >= vs2)	(Color >= i), (Shape = round), (Price < 2000), (Carat = 1), (Clarity >= vs2)

Total:

---	PRECISION	RECALL
PROJECT	0.88	0.91
SELECT	0.88	0.77

projection Correct: 134
 projection Attempted: 153
 projection Should have been: 147
 selection Correct: 105
 selection Attempted: 120
 selection Should have been: 137