Computing Intensional Answers to Questions: An Inductive Logic Programming Approach

Philipp Cimiano[a], Sebastian Rudolph[b], Helena Hartfiel[c]

[a] *Web Information Systems, Delft University of Technology, 2600 GA Delft, The Netherlands*
[b] *Institute AIFB, Universität Karlsruhe (TH), D-76131 Karlsruhe, Germany*
[c] *disy Informationssysteme GmbH, D-76133 Karlsruhe, Germany*

Abstract

Research on natural language interfaces has mainly concentrated on question interpretation as well as answer computation, but not focused as much on answer presentation. In most natural language interfaces, answers are in fact provided extensionally as a list of all those instances satisfying the query description. In this paper, we aim to go beyond such a mere listing of facts and move towards producing additional descriptions of the query results referred to as 'intensional answers'. We define an intensional answer (IA) as a logical description of the actual set of answer items to a given query in terms of properties that are shared by exactly these answer items. We argue that IAs can enhance a user's understanding of the answer itself but also of the underlying knowledge base. In particular, we present an approach for computing an intensional answer given an extensional answer (i.e. a set of entities) returned as a result of a question. In our approach, an intensional answer is represented by a clause and computed based on Inductive Logic Programming (ILP) techniques, in particular bottom-up clause generalization. The approach is evaluated in terms of usefulness and time performance, and we discuss its potential for helping to detect flaws in the knowledge base as well as to interactively enrich it with new knowledge. While the approach is used in the context of a natural language question answering system in our settings, it clearly has applications beyond, e.g. in the context of research on generating referring expressions.

Key words: natural language interfaces, intensional answers, applications of inductive logic programming

1. Introduction

Question Answering systems for unstructured ([1]) or structured information ([2]) - typically referred to as Natural Language Interfaces (NLIs) in the latter case - have been a focus of research since a few decades now (see [3] and [4] for somewhat older overviews of NLI research). They are a crucial component towards providing users with intuitive access to the vast amount of information available world-wide in the form of resources as heterogeneous as web sites, classical and Semantic Web databases, RSS feeds, blogs, wikis, etc.

However, most of the prevalent work has concentrated on providing *extensional* answers to questions. In essence, what we mean by an extensional answer here is a list of instances that satisfy the query[1]. For example, a question like: *Which states have a capital?*, when asked to a knowledge base about Germany would deliver an (extensional) answer consisting of the 16 federal states ("Bundesländer"): *Baden-Württemberg, Bayern, Rheinland-Pfalz, Saarland, Hessen, Nordrhein-Westfalen, Niedersachsen, Bremen, Hamburg, Sachsen, Brandenburg, Sachsen-Anhalt, Mecklenburg-Vorpommern, Schleswig-Holstein, Berlin,* and *Thüringen.* While this is definitely a correct answer, it is certainly not maximally informative. An arguably more informative answer would—beyond a mere listing of the matching instances—also *describe* the answers in terms of relationships which can help a user to better understand the answer space. To a question *"Which states have a capital?"* a system could, besides delivering the full extension, also return an answer such as *"All states (have a capital.)"*

The predominant view in Question Answering research is that answers are essentially lists of answer items. In fact, there has not been much research on aspects of answer presentation, in particular on:

- integrating additional external data for presentation and visualization purposes (e.g. mashups),
- structuring the answer space, i.e. by grouping/clustering answers,
- constructing explanations or proofs for answers,

[1] Actually, the bindings of the query variables in a query (seen as a logical formula) to instances.

- providing partial answers, and
- delivering non-extensional (e.g. intensional) answers.

While all above mentioned aspects are worthwhile to be investigated further, this paper will be focused on the last point. We will in fact refer to such descriptions which go beyond the mere enumeration of the answer items as *intensional answers* (IAs). The intension in this sense is thus a description of the answer items in terms of a *distinguishing* set of properties *common* to all of them. In other words, the description is satisfied by all answer items but by no item that is not contained in the answer set. In the above case, the common property of the answers is that they represent exactly the set of all federal states in the knowledge base.

It could be certainly argued that providing intensional descriptions "over-answers" the question.

As a counterargument to this objection it is important to mention that, first of all, the extension might simply be too large for a user to grasp it completely. In this case, a compact representation of the answer in terms of an intensional description might be useful.

Second, an intensional answer can indeed provide new but relevant knowledge to the user. For example, let us consider the question *"Which states does the Spree flow through?"*, with answers: *Berlin* and *Brandenburg*. The intensional answer to this question, as generated by our approach, is: *'(By the way,) All the states which the Havel flows through.'* From this answer, the user learns that the Havel and the Spree flow through exactly the same states and this that the Havel also flows through Berlin and Brandenburg.

Third and most importantly, as we will see in the remainder of this paper, IAs can help to detect flaws in the knowledge base as well as suggest refinements. In the case of the above mentioned question *"Which states have a capital?"*, the user could be asked whether to add the axiom that *"All states have a capital."* into the knowledge base, thus refining it in a query-driven way. The confirmation by a user would be necessary as the axiom has been derived inductively and it is not clear if it holds universally or only in the given state of the knowledge base.

We present an approach for computing intensions of queries on the basis of their extensions and a given knowledge base. Our research has been performed in the context of the natural language interface ORAKEL (see

[2]) and the aim has been to integrate the component for providing intensional answers into the system. Notwithstanding, the approach presented here could be used in any setting where intensional answers are appropriate (and as we will see in the related work section it has strong connections to research on generating referring expressions). We describe an approach based on Inductive Logic Programming (ILP) and in particular based on bottom-up clause generalization which computes a clause covering the extension of the query and thus representing an intensional answer in the sense that it describes it in terms of features which only the elements in the extension have in common. In particular, the system iteratively calculates least general generalizations (LGGs) for the answers by adding one answer item at a time and generalizing the clause computed so far.

The paper is structured as follows: in Section 2, we describe the ORAKEL system in more detail in order to provide the reader with some background of our research. We present the approach for generating intensional answers in Section 3. In Section 4, we describe the empirical evaluation of our approach, which has been carried out based on the dataset previously used in the evaluation of the ORAKEL system (see [2]). We analyze in detail to what extent the intensional answers produced by the system are "useful" and also discuss how the intensional answers can help in detecting errors in the knowledge base. We discuss related work in Section 5 and conclude in Section 6.

2. Background: The ORAKEL System

The ORAKEL system is a portable and compositional natural language interface to knowledge bases. It is portable as it can be easily adapted to new ontologies and knowledge bases by non-experts. This has been corroborated before by experimental data (see [2]). It is compositional in the sense that it interprets the question by constructing a logical formula representing the question, accomplishing this in a compositional way, i.e. on the basis of the "meaning" of the constituents of the question and the way they are (syntactically) combined (as made explicit by a syntactic analysis of the sentence).

The main components of the system as well as their interaction are depicted in Figure 1. These components are:

- **Query Interpreter**: translating the user question into a generic logical form,

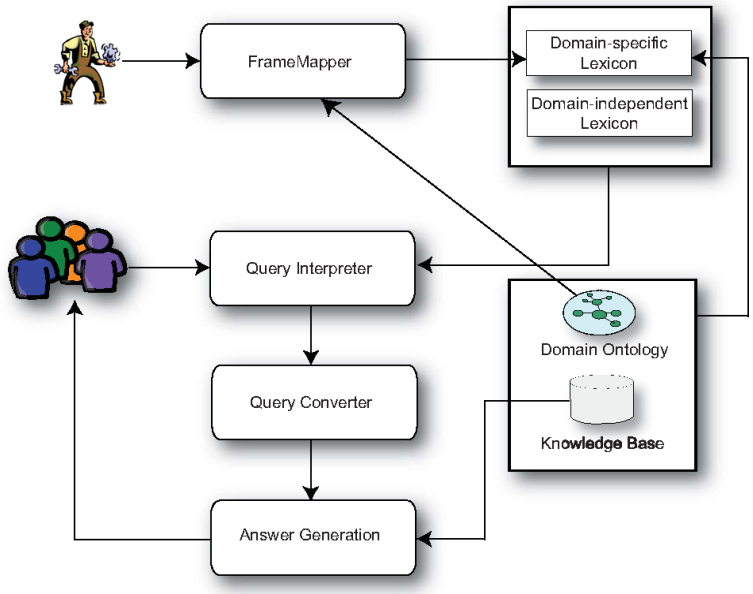

Figure 1: Overview of the ORAKEL system

- **Query Converter**: converting the generic logical form into the target query language,
- **Answer Generation**: the component responsible for evaluating the query (w.r.t. the knowledge base) and presenting the answers to the user (as a list of instances satisfying the query so far),
- **FrameMapper**: a graphical user interface used to adapt the system to a given domain by creating an ontology-specific lexicon,
- **Domain-independent Lexicon**: a lexicon specifying the domain-independent meaning of closed class words (articles, wh-pronouns, prepositions etc.), and the
- **Domain-specific Lexicon**: a lexicon generated specifically for the ontology in question.

The underlying syntactic theory of our system is a formalism called Logical Description Grammars (LDGs, compare [5]), which is inspired by Lexi-

calized Tree Adjoining Grammars (LTAGs, see [6]). The structures used in LDG are essentially (descriptions of) trees consisting of nodes labeled with syntactic information (compare Figure 2). Positive nodes in essence provide arguments while negatively marked nodes require some arguments to be inserted. An important characteristic of these trees is that they encapsulate all syntactic/semantic arguments of a word. In the LDG formalism used in ORAKEL, there is only one syntactic composition operation, which consists in identifying positively and negatively marked nodes with each other within one or across trees. Hereby, two nodes can only be identified with each other if (i) they have complementary marks (negative/positive), (ii) they have the same syntactic category, (iii) their feature structures are compatible as well as (iv) syntactic dominance and surface order of words is respected. The parser of the ORAKEL system essentially thus plugs negatively and positively marked nodes together (see [7]).

As formalism for semantic composition, the lambda calculus is used, relying on functional abstraction, functional application (β-conversion) and variable renaming (α-conversion) to compose the meanings of the various question parts into a complete logical query (compare [8]).

Figure 2 shows the parse tree, decorated with lambda expressions conveying the semantics for the question *Which river passes through Berlin?*

The meaning of the diverse lexico-syntactic units in the input can be expressed in functional lambda notation roughly[2] as follows:

Which river	$\lambda P\ ?x\ (river(x) \wedge P(x))$
passes through	$\lambda x\, \lambda y\ \mathit{flow\_through}(x, y)$
Berlin	$\lambda Q\ Q(Berlin)$

While *passes through* is interpreted here as referring to the `flow_through` relation, this is obviously not the case in general, as it can also refer to the relation `located_at_highway` (with inverted arguments) modeling which highways pass by which cities. The two different 'readings' of *pass through* can be distinguished by taking into account restrictions on their slots, i.e. rivers in the former and highways in the latter case. This is the reason why the right interpretation of *pass through* (i.e. `flow_through`) can be selected

[2]Roughly as in principle each word should be associated with a semantic representation. We abstract from this for the sake of clarity of presentation.

in the example question. The semantic representation of *'passes through'* expects two individuals as arguments to be inserted into the appropriate relation `flow_through`. The expression *'which river'* expects some property P which x, a river, needs to satisfy. *'Berlin'* requires some predicate Q into which it can be inserted as an argument.

Given the simplified syntactic structure together with instructions on how the semantic expressions are applied to each other in Figure 2, and evaluating the tree in a standard bottom-up fashion, we would first carry out the functional application

$$\lambda u\ (\lambda Q\ Q(Berlin))((\lambda x\, \lambda y\ \mathit{flow\_through}(x, y))(u)),$$

yielding as semantic representation of the VP node

$$\lambda u\ \mathit{flow\_through}(u, Berlin)$$

in which the argument *Berlin* has been correctly inserted. To obtain the final semantic representation of the top sentence node S, we would carry out the functional application

$$(\lambda P\, ?x\, (river(x) \wedge P(x)))(\lambda u\ \mathit{flow\_through}(u, Berlin)),$$

resulting in the final logical query:

$$?x\ (river(x) \wedge \mathit{flow\_through}(x, Berlin))$$

After the question has been translated into logical form, the *Query Converter* is invoked to translate this question into the target query language of our choice. The above query would be represented in the F-Logic query language [9] as follows:

$$FORALL\ X \leftarrow X : river\ AND\ X[flow\_through \rightarrow Berlin].$$

The answer to this query is then generated by evaluating it with respect to the inference engine (in this case the OntoBroker system [10]), yielding the following answers: Havel, Spree.

The ORAKEL system provides the flexibility to be applied to various knowledge representation paradigms (see [2]), i.e. it has been applied to OWL [11] and F-Logic [9], and supports any reasonably expressive query language.

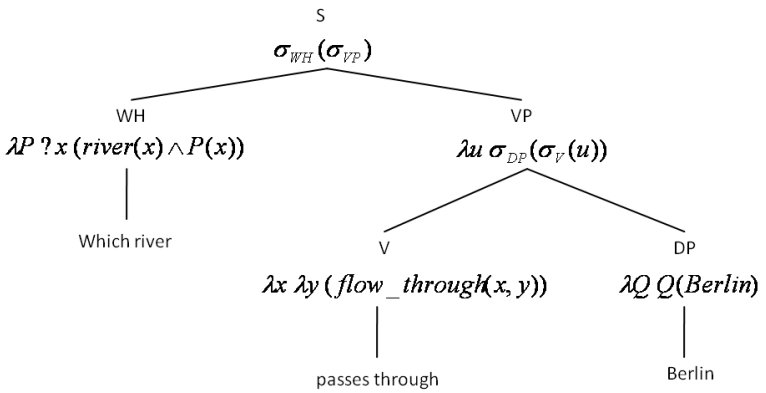

Figure 2: Syntactic analysis with semantic representations for each word specified according to the λ-calculus and instructions on how to combine the different representations with each other.

If the query language is not expressive enough, then some of the queries interpreted by the ORAKEL system will not be translated into the target language. If the query language is too expressive this does not constitute a problem for the ORAKEL system. In order to exchange the query language, only a simple query rewriting interface needs to be instantiated. In this work, the implementation is based on F-Logic [9], and Ontobroker [10] is used as the underlying inference engine.

The results of the ORAKEL system have been shown to be reasonable and competitive in comparison to other state-of-the-art approaches, yielding a Precision of 84.23% and a Recall of 53.67% on the dataset described in [2]. For more details about the ORAKEL system, the interested reader is referred to [2].

3. Approach

In this section, we first present an overview of the component for generating intensional answers (Section 3.1), then we introduce the basis for the computation of least general generalizations (LGG) of clauses in Section 3.2 and present the generalization algorithm in more detail in Section 3.3. The crucial reduction step, yielding more compact clauses, is presented in Section 3.4. Section 3.5 presents a worked example. We conclude this section with a short note on the worst-case complexity and the scalability of the approach.

3.1. Overview

The aim of the work described in this article has been to extend the ORAKEL natural language interface with capabilities for delivering intensional answers. The workflow of the system is depicted in Figure 3. When the user asks a natural language question w.r.t. the knowledge base, it is processed by the ORAKEL system that computes a logical query which can then be evaluated by the inference engine with respect to the KB and ontology. With KB we mean here a set of ground facts, i.e. instances, while with ontology we refer to a logical theory containing axioms constraining the set of possible models according to a given conceptualization [12]).

While the extensional answer is displayed to the user, it is also forwarded to the component which generates intensional answers (named *Bottom-up Generalization* in the figure). By means of an ILP-algorithm, this component provides a hypothesis (in the form of a clause) based on the extensional answer. This hypothesis is the intensional answer and is displayed in addition to the extensional answer. Note that in this paper, we are not concerned with the problem of generating an adequate answer in natural language, but rather with the previous step, i.e. the computation of a clause which represents the intensional answer. Obviously, natural language generation techniques can then be applied to generate an answer in natural language. However, this is not the focus of the present paper, but an obvious direction for future work.

We assume that even in the case of a useful hypothesis, the user is interested in the extensional answer as well, such that we do not forego its presentation. The answer items are presented in parallel to the generation of their intensional description, such that the user does not have to wait for them until an intensional answer is possibly generated.

Before introducing the technical details of our approach, we briefly introduce the basics of the least general generalization (LGG) of clauses in the next section.

3.2. Basics: LGG Computation

Our approach to computing intensional answers is based on bottom-up clause generalization, which computes a clause subsuming the extension of an answer, thus representing an intensional answer in the sense that it describes it in terms of properties which all elements in the extension (and only these) share. In particular, the system iteratively calculates least general generalizations (LGGs) ([13] and below) for the answers by adding one answer at a time and generalizing the clause computed so far. According to our

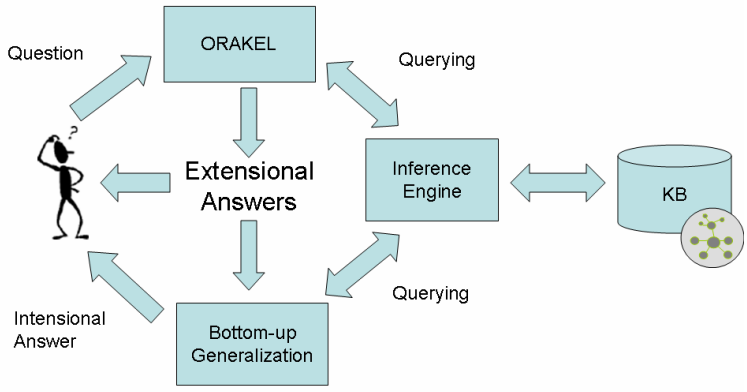

Figure 3: Workflow of the system

formulation as an ILP problem, the intensional description of the answers is regarded as a hypothesis which covers all the positive answers and is to be found in a search space of definite Horn clauses. This search space of definite Horn clauses is structured by a relation called θ-subsumption, which orders the hypotheses (clauses) in a lattice, thus allowing to effectively navigate the search space (compare [13], pp. 33-37).

Let us first recap some basic definitions needed for describing the computation of LGGs. Thereby we assume some basic familiarity with first order predicate logic.

Definition 1 (Clauses)
A clause *is a first order formula of the form $L_1 \vee \ldots \vee L_n$, where L_i are positive or negative literals. For easier representation, clauses are usually conceived as sets $\{L_1, \ldots, L_n\}$. A positive literal is an atom $p(t_1, \ldots, t_k)$, a negative literal is a negated atom $\neg p(t_1, \ldots, t_k)$, where p is a predicate of arity k and $t_1, \ldots, t_k$ are terms.*

As we will deal with a function-free variant of first order logic, the only terms we will encounter are constants and variables.

The clauses we will be concerned with in this paper are of a special type, namely *definite Horn clauses*. These are clauses with exactly one positive literal (and arbitrarily many negative ones). They can be equivalently represented as implications (also denoted as rules) of the form $A \leftarrow A_1 \wedge \ldots \wedge A_k$, where $A, A_1, \ldots, A_k$ are atoms. Moreover, A is then referred to as the head

of the rule and $A_1 \wedge \ldots \wedge A_k$ as its body.

Definition 2 (Theta-subsumption)
A substitution θ is a function mapping variables to terms. Given a clause c, the clause $c\theta$ is obtained by substituting the variables occurring in the literals contained in c according to θ.

We then say that c θ-subsumes c' iff there exists a substitution θ such that $c\theta \subseteq c'$.

θ-subsumption induces a partial order between clauses. In particular, each pair of clauses has an upper bound in the generalization lattice. The partial order $\leq$ is defined by: $c \leq c'$ iff c θ-subsumes c'. In that case we can also say that c is at least as general as c'. Note that $c \leq c'$ always implies that c' is a logical consequence of c ($c \models c'$) whereas the converse does not hold (see [13]).

θ-subsumption allows to prune large parts of the search space as if we generalize a clause c' to c, all the examples covered by c' will also be covered by c. Consequently if c' covers a negative example, all of its generalizations will also cover the negative example, such that we do not have to examine the generalizations of c' further. We refer to the least upper bond of two clauses c and c' in the generalization lattice as the least general generalization (LGG) of c and c'.

The LGG of two clauses is computed in essence as described in [13], pp. 41:

Definition 3 (LGG)

$$LGG(c_1, c_2) := \bigcup_{L_i \in c_1, L_j \in c_2} LGG(L_i, L_j)$$

where the LGG of literals is defined as follows:

- *If both literals are positive:*
  $LGG(p(t_1, ..., t_n), p(s_1, ..., s_n)) = \{p(LGG(t_1, s_1), ..., LGG(t_n, s_n))\}$
  $LGG(p(t_1, ...t_n), q(s_1, ..., s_n)) = \emptyset$ *in case* $p \neq q$.

- *If they are negative:*
  $LGG(L_i, L_j) = LGG(\neg A_i, \neg A_j) = \neg LGG(A_i, A_j)$.

- *Otherwise, if one is positive and one is negative:*
  $LGG(L_i, L_j) = \emptyset$.

For terms, the LGG is defined as follows:

- $LGG(t, t) = t$
- $LGG(s, t) = V$ *where* $s \neq t$.

Thereby, V *is a variable not occurring in* c_1 *or* c_2*, which is uniformly used to represent the LGG of* s *and* t.

While there is explicit negation in the F-Logic language and this is supported by the F-Logic implementation of OntoBroker, we do only use definite Horn clauses so far. However, the above definition is general enough to accommodate negation in future stages. Still, it is easy to see that when applied to two definite Horn clauses having the same head predicate, the resulting clause is again definite and Horn and also carries the same head predicate.

From the way the LGG is constructed, it can be shown that it is the most specific clause (w.r.t. $\leq$) such that: $LGG(c_1, c_2) \leq c_1$ and $LGG(c_1, c_2) \leq c_2$.

While F-Logic and OntoBroker support function symbols, our assumption here is that the knowledge base is actually function-free, so that we do not support function symbols in our approach to generating an intensional description.

3.3. Generalization

In order to compute a clause which covers all the extensional answers, we iterate over all the answers and compute pairwise LGGs by adding one answer at a time. We make sure that the resulting clause is consistent (in the sense that it does not cover negative examples) by evaluating it with respect to the knowledge base. Thereby, we rely on the inference engine in order to check that no other than the original answer items are returned. In fact, any definite Horn clause with an artificial *answer*-predicate in the head can be easily translated into a corresponding query to the knowledge base. Our learning algorithm is thus similar to the classical FindS algorithm described in [14] in the sense that it aims at finding one single hypothesis which covers all positive examples and ignores negative examples during learning. If the resulting hypothesis which covers all positive examples is consistent, then an appropriate clause has been found. Otherwise, there is no clause which covers the positive examples without overgeneralizing.

In our approach, we perform the search of the hypothesis space in a bottom-up manner. We start with the most specific hypothesis which covers

```
IntensionalAnswerClause(Set Answers, KnowledgeBase KB)
 1  a = Answers.getNext();
 2  c = constructClause(a, KB);
 3  while not all answers have been processed
 4  do
 5      a' = Answers.getNext();
 6      c' = constructClause(a', KB);
 7      c = LGG(c, c');
 8      Answers' = evaluateQuery(query(c), KB);
 9      if Answers ∩ Answers' ⊂ Answers'(i.e. c is inconsistent)
10        then
11              return ∅ ( no consistent clause can be found )
12
13      c'' = reduceClause(c, KB, Answers)
14      Answers'' = evaluateQuery(query(c''), KB);
15      if Answers'' = Answers (i.e. c'' covers all answers)
16        then
17              return c'';
18
19      Answers = Answers\Answers' (optimization)
20  return reduceClause(c, KB, Answers);
```

Figure 4: Generalization Algorithm (calculating a reduced clause after each LGG computation)

a given example and generalize this clause with the specific hypothesis for the next example by means of the LGG until it can not be further generalized without covering negative examples. Our approach proposes an efficient and simple way to generate an intensional answer as we do not construct the hypothesis from scratch, but generalize specific hypotheses representing one element in the extension of the answer.

The generalization algorithm is given in Figure 4. The algorithm takes as input the set of (extensional) answers (*Answers*) as well as the knowledge base $\mathcal{KB}$. The aim is to generate a hypothesis which exactly covers the extensional answers. A hypothesis in our approach is actually a definite Horn clause. For the hypothesis generation, the ILP-learner first constructs a specialized clause based on the first answer – or positive example from the

ILP point of view (line 1). This is achieved by sending queries asking for the names of methods or concepts[3] containing the constant representing the answer as argument. The received answers are used to define literals which, arranged in the body of a definite Horn clause, represent the intension of the first example. As head of this clause, we use an artificial target predicate *answer*.[4] The *constructClause*$(a, \mathcal{KB})$ procedure (line 2) returns such a clause for a certain individual a in the knowledge base $\mathcal{KB}$ consisting of all the factual information directly referring to a that can be retrieved from the knowledge base.

In our example, the first answer (positive example) to the query is: *Saarland* and the following specialized clause for this example is generated:

$$
\begin{aligned}
c := answer(Saarland) \quad \leftarrow \quad & state(Saarland), location(Saarland), \\
& inhabitants(Saarland, 1.062.754), \\
& borders(Saarland, Rheinland\text{-}Pfalz), \\
& borders(Saarland, France), \\
& borders(Saarland, Luxembourg), \\
& borders(Rheinland\text{-}Pfalz, Saarland), \\
& borders(France, Saarland), \\
& borders(Luxembourg, Saarland), \\
& flows\_through(Saar, Saarland), \\
& location(Saarbrücken, Saarland), \\
& capital\_of(Saarbrücken, Saarland).
\end{aligned}
$$

If there is only one answer, the constructed clause will be exactly the intensional answer we are searching for. In such a case, the user thus obtains additional information about the entity in terms of the concepts it belongs to as well as other entities it is related to. In case there are more answers (line 3), we loop over these and compute the LGG of the clause c constructed so far and the clause c' (line 7) constructed on the basis of the next answer a' (lines 5 and 6).

In our example, the next specialized clause c' for the next answer *Mecklenburg-Vorpommern* is:

[3]Methods and concepts are the F-Logic notions for binary and unary predicates, respectively.

[4]Essentially, the target predicate is used to indicate which variable is the one actually referring to the instances that have been asked for. In that sense, it resembles the SELECT statement in SQL.

$$
\begin{aligned}
c' := answer(Mecklenburg\text{-}Vorp.) \leftarrow\ & state(Mecklenburg\text{-}Vorp.),\\
& location(Mecklenburg\text{-}Vorp.),\\
& inhabitants(Mecklenburg\text{-}Vorp., 1.735.000),\\
& borders(Mecklenburg\text{-}Vorp., Brandenburg),\\
& borders(Mecklenburg\text{-}Vorp., Niedersachsen),\\
& borders(Mecklenburg\text{-}Vorp., Sachsen\text{-}Anhalt),\\
& borders(Mecklenburg\text{-}Vorp., Schleswig\text{-}Holstein),\\
& borders(Brandenburg, Mecklenburg\text{-}Vorp.),\\
& borders(Niedersachsen, Mecklenburg\text{-}Vorp.),\\
& borders(Sachsen\text{-}Anhalt, Mecklenburg\text{-}Vorp.),\\
& borders(Schleswig\text{-}Holstein, Mecklenburg\text{-}Vorp.),\\
& location(Rostock, Mecklenburg\text{-}Vorp.),\\
& location(Schwerin, Mecklenburg\text{-}Vorp.),\\
& capital\_of(Schwerin, Mecklenburg\text{-}Vorp.).
\end{aligned}
$$

Computing the LGG of c and c', we obtain: $LGG(c, c') :=$

$$
\begin{aligned}
answer(X) \leftarrow\ & state(X), location(X), inhabitants(X, Y),\\
& borders(X, V_{1,1}), borders(X, V_{1,2}), borders(X, V_{1,3}), borders(X, V_{1,4}),\\
& borders(V_{1,1}, X), borders(V_{1,2}, X), borders(V_{1,3}, X), borders(V_{1,4}, X),\\
& borders(X, V_{2,1}), borders(X, V_{2,2}), borders(X, V_{2,3}), borders(X, V_{2,4}),\\
& borders(V_{2,1}, X), borders(V_{2,2}, X), borders(V_{2,3}, X), borders(V_{2,4}, X),\\
& borders(X, V_{3,1}), borders(X, V_{3,2}), borders(X, V_{3,3}), borders(X, V_{3,4}),\\
& borders(V_{3,1}, X), borders(V_{3,2}, X), borders(V_{3,3}, X), borders(V_{3,4}, X),\\
& borders(V_{*,5}, V_{1,*}), borders(V_{*,6}, V_{1,*}), borders(V_{*,7}, V_{1,*}), borders(V_{*,8}, V_{1,*}),\\
& borders(V_{*,5}, V_{2,*}), borders(V_{*,6}, V_{2,*}), borders(V_{*,7}, V_{2,*}), borders(V_{*,8}, V_{2,*}),\\
& borders(V_{*,5}, V_{3,*}), borders(V_{*,6}, V_{3,*}), borders(V_{*,7}, V_{3,*}), borders(V_{*,8}, V_{3,*}),\\
& borders(V_{4,*}, V_{*,1}), borders(V_{4,*}, V_{*,2}), borders(V_{4,*}, V_{*,3}), borders(V_{4,*}, V_{*,4}),\\
& borders(V_{5,*}, V_{*,1}), borders(V_{5,*}, V_{*,2}), borders(V_{5,*}, V_{*,3}), borders(V_{5,*}, V_{*,4}),\\
& borders(V_{6,*}, V_{*,1}), borders(V_{6,*}, V_{*,2}), borders(V_{6,*}, V_{*,3}), borders(V_{6,*}, V_{*,4}),\\
& location(P, X), location(Q, X), capital\_of(Q, X).
\end{aligned}
$$

The variables above then represent the LGGs of the following pairs of terms in the original clauses:

Variable	Term c	Term c'
X	Saarland	Mecklenburg-Vorpommern
Y	1.062.754	1.735.000
$V_{1,1}$	Rheinland-Pfalz	Brandenburg
$V_{1,2}$	Rheinland-Pfalz	Niedersachsen
$V_{1,3}$	Rheinland-Pfalz	Sachsen-Anhalt
$V_{1,4}$	Rheinland-Pfalz	Schleswig-Holstein
$V_{2,1}$	France	Brandenburg
$V_{2,2}$	France	Niedersachsen
	...	...
$V_{3,4}$	Luxembourg	Schleswig-Holstein
$V_{1,*}$	Rheinland-Pfalz	Mecklenburg-Vorpommern
$V_{2,*}$	France	Mecklenburg-Vorpommern
$V_{3,*}$	Luxembourg	Mecklenburg-Vorpommern
$V_{4,*}$	Rheinland Pfalz	Mecklenburg-Vorpommern
$V_{5,*}$	France	Mecklenburg-Vorpommern
$V_{6,*}$	Luxembourg	Mecklenburg-Vorpommern
$V'_{*,1}$	Saarland	Brandenburg
$V'_{*,2}$	Saarland	Niedersachsen
$V'_{*,3}$	Saarland	Sachsen-Anhalt
$V'_{*,4}$	Saarland	Schleswig-Holstein
$V'_{*,5}$	Saarland	Brandenburg
$V'_{*,6}$	Saarland	Niedersachsen
$V'_{*,7}$	Saarland	Sachsen-Anhalt
$V'_{*,8}$	Saarland	Schleswig-Holstein
P	Saarbrücken	Rostock
Q	Saarbrücken	Schwerin

The resulting clause is then transformed into a query in the target language, i.e. F-Logic in our case, so that it can be evaluated by the inference engine, which returns the extension of the clause $LGG(c, c')$ on the basis of the given knowledge base (line 8). If the clause is inconsistent (line 9), i.e., it covers more answers than the required ones (this is the case if $Answers \cap Answers' \subset Answers'$) then no consistent clause can be constructed and the algorithm returns the empty clause $\emptyset$ (line 11).

As a further optimization of the algorithm, we could always remove those examples which have been already covered by the clause computed so far (see line 19). However, we have not made use of this optimization in our experimental settings.

Note that the clause computed as the LGG can grow exponentially in the size of the original clauses. However, the clauses representing the LGG can also be reduced. In case the clause c is consistent, it is reduced as

described below in Section 3.4 (line 13) and we test if this reduced clause covers exactly the original answers (line 15). If so, we return the reduced clause c'' and are done (line 17). Otherwise, we proceed to consider the next answer and compute the next least general generalization between the last unreduced clause c and the clause c' generated by the next answer a' (lines 5-7). This algorithm returns a clause in case it covers exactly the set of extensional answers.

By removing redundant literals, the above LGG can be reduced to:

$$\begin{aligned} answer(X) \leftarrow\ & state(X), location(X), inhabitants(X,Y), borders(X,Z), \\ & borders(Z,X), location(P,X), capital\_of(Q,X) \end{aligned}$$

The above clause is intensionally equivalent to the LGG, i.e. the above clause and the LGG θ-subsume each other. All atoms of the form $borders(X,*)$ and $borders(*,X)$ (12 each) are intensionally equivalent to the above atoms $borders(X,Z)$ and $borders(Z,X)$. All the other border-predicates can be removed as they are trivially fulfilled in case there is at least one instance of the border-predicate. This is certainly the case in our LGGs as the clauses in the LGG appear only in case they appear in one specialized clause, such that it is guaranteed that there is at least one instance.

The above clause can be transformed into a query as follows and sent to the inference engine to be evaluated, yielding the answers below:

$$\begin{aligned} FORALL\ X \leftarrow\ & EXISTS\ Y,Z,P,Q\ X : state\ AND\ X : location\ AND \\ & X[inhabitants \rightarrow Y]\ AND\ X[borders \rightarrow Z]\ AND \\ & Z[borders \rightarrow X]\ AND\ P[location \rightarrow X]\ AND \\ & Q[capital\_of \rightarrow X]. \end{aligned}$$

Nordrhein-Westfalen
Bayern
Hessen
Baden-Württemberg
Berlin (Bundesland)
Brandenburg (Bundesland)
Niedersachsen
Sachsen
Sachsen-Anhalt
Thüringen
Schleswig-Holstein
Hamburg (Bundesland)
Rheinland-Pfalz
Saarland
Mecklenburg-Vorpommern

Note that the state of *Bremen* is missing from the above list. The reason is that Bremen is not modeled as bordering any state, such that the atoms $borders(X, Z)$ and $borders(X, Z)$ are not satisfied.

Thus, the clause reduction we discuss below accomplishes two goals: i) it makes the intensional answer more compact and ii) it potentially increases the coverage (in terms of positive answers) of the query by removing certain literals.

Concerning the first case, assuming that the above clause would cover all of the examples in the original answers, it is highly redundant from the point of view of the knowledge base. For example, any state X is also a location according to our ontology (as *state* is subsumed by *location* in our ontology). Further, every state can be assumed to have some number of inhabitants Y as well as some capital Q, etc. Thus, all the literals besides $state(X)$ are irrelevant in the sense that removing them will not introduce negative examples but potentially cover more positives (like the missing *Bremen* in our case). In order to obtain more compact (but consistent) descriptions, we apply a clause reduction step which we describe in the next section.

3.4. Clause Reduction

The clause reduction algorithm removes body literals in order to potentially increase the coverage of the clause in terms of positive examples without introducing any negative ones. Further, the reduction yields a more compact

clause which might be more intuitive and informative for the user of the system compared to a longer clause.

Note that by removing literals we create a clause which is at least as general as the original clause, so that the number of examples can increase but not decrease. Thus, it is important to verify that no negative examples are covered by the resulting clause, i.e. that it is consistent.

The following algorithm performs the reduction of the clause:

```
REDUCECLAUSE( Clause c, KnowledgeBase KB, Set Answers)
 1  List literals = orderLiterals(c); (in increasing order)
 2  for i = 1 to |literals|
 3      do
 4          c' = remove(c, L_i);
 5          Answers' = evaluateQuery(query(c'), KB);
 6          if Answers ∩ Answers' = Answers' (i.e. c' is consistent)
 7             then
 8                    c = c'
 9
10  return c;
```

The algorithm first sorts the literals in the clause according to a specific total order given below (line 1) and checks incrementally (line 2) whether each of these literals (following the ordered list) can be removed without covering any negative examples. It does so by constructing a query from the clause c' with the removed literal (line 4) and evaluating this query with respect to the knowledge base (line 5). If c' is consistent (line 6), the change is applied to the clause (line 8). As result, the algorithm returns a clause which is at most as long as the original query. In case no literal can be removed while preserving consistency at the same time, the original clause is returned.

The literals are ordered according to a total order satisfying the following conditions:

- $L_i < L_j$ if L_i is an atom of higher arity than L_j
- $L_i < L_j$ if L_i and L_j have the same arity and less variables in L_i appear in the remaining literals of the clause.

For the literals in the following clause:

$$answer(X) \leftarrow state(X), location(X), inhabitants(X,Y), borders(X,Z), \\ borders(Z,X), location(P,X), capital\_of(Q,X)$$

the following order would be admissible:

$$location(P,X) < capital\_of(Q,X) < inhabitants(X,Y) < \\ borders(X,Z) < borders(Z,X) < location(X) < state(X)$$

Thus, we would first attempt to remove the literals $location(P,X)$ and $capital\_of(Q,X)$ and $inhabitants(X,Y)$, as they have the highest arity and the variables P, Q and Y only occur once. The rationale for eliminating such literals should be clear: as they share fewer variables with other literals, their effect on the other literals (in terms of dependencies) is smaller. Then we would remove $borders(X,Z)$ and $borders(Z,X)$ as they share two variables with other literals in the body of the query. Finally, we would consider the literals with an arity of 1 for removal, i.e. $location(X)$ and $state(X)$. This order criterion of descending predicate arity was chosen in order to obtain descriptions that are preferably intuitive.

In fact, we have implemented two different procedures for the elimination of irrelevant literals. These procedures differ with respect to whether the clause is reduced after the computation of each LGG (as sketched in the algorithm depicted in Figure 4, line 13) or only at the end. Reduction at the end is unproblematic as the clause then already covers all the positive examples and the reduction does not affect the coverage. In case the reduction is performed after each LGG-computation, this can affect the computation of the further LGGs. Therefore, in the version of our approach in which we compute a reduced clause after each iteration, the reduced clause is only used as output in case it consistently covers all answers. Otherwise, the unreduced clause is used in the next iteration. Our experiments actually show that by applying the reduction procedure after each iteration we can reduce the number of iterations and speed up our algorithm in many cases.

3.5. A Worked Example

After having explained the different steps involved in the computation of an intensional answer, we now discuss the computation of an intensional

description for our running example in an end-to-end fashion, abstracting from details explained in earlier sections to enhance the overall understanding of the approach.

Let's consider our running example, the question: *"Which states have a capital?"*. ORAKEL translates this question into the following logical query: $FORALL\ X \leftarrow EXISTS\ Y\ X : state \wedge X[capital \rightarrow Y]$. The answer of the ORAKEL system can be represented as follows in clause notation:

answer(Saarland)
answer(Mecklenburg-Vorpommern)
answer(Rheinland-Pfalz)
answer(Hamburg (Bundesland))
answer(Schleswig-Holstein)
answer(Thüringen)
answer(Sachsen-Anhalt)
answer(Sachsen)
answer(Bremen)
answer(Niedersachsen)
answer(Brandenburg)
answer(Berlin (Bundesland))
answer(Baden-Württemberg)
answer(Hessen)
answer(Bayern)
answer(Nordrhein-Westfalen)

Now the goal of our approach is to find a clause describing the *answer*-predicate intensionally. The number of positive examples is 16, while there are no explicit negative examples given. Nevertheless, a clause is inconsistent if, evaluated with respect to the knowledge base by the inference engine, it returns answers which are not in the set of positive examples. This is verified by the condition $Answers \cap Answers' \subset Answers'$ in the algorithm from Fig. 4.

In what follows, we illustrate our algorithm using the version which reduces the learned clause after each iteration. The first example considered is *"Saarland"*. The algorithm is initialized with the following clause representing all the facts about *"Saarland"* in the knowledge base (compare Section 3.2):

$$
\begin{aligned}
answer(Saarland) \leftarrow\ & state(Saarland), location(Saarland), \\
& inhabitants(Saarland, 1.062.754), borders(Saarland, Rheinland\text{-}Pfalz), \\
& borders(Saarland, France), borders(Saarland, Luxembourg), \\
& borders(Rheinland\text{-}Pfalz, Saarland), location(Saarbrücken, Saarland), \\
& capital\_of(Saarbrücken, Saarland), borders(France, Saarland), \\
& borders(Luxembourg, Saarland), flows\_through(Saar, Saarland)
\end{aligned}
$$

The above specialized clause with the artificial predicate *answer* as head is produced by constructClause("Saarland"). As there are more examples, the next example *"Mecklenburg-Vorpommern"* is considered, which is represented by the following clause (see Section 3.2).

$$
\begin{aligned}
answer(Mecklenburg\text{-}Vorpommern) \leftarrow\ & state(Mecklenburg\text{-}Vorpommern), \\
& location(Mecklenburg\text{-}Vorpommern), \\
& inhabitants(Mecklenburg\text{-}Vorpommern, 1.735.000), \\
& borders(Mecklenburg\text{-}Vorpommern, Brandenburg), \\
& borders(Mecklenburg\text{-}Vorpommern, Niedersachsen), \\
& borders(Mecklenburg\text{-}Vorpommern, Sachsen\text{-}Anhalt), \\
& borders(Mecklenburg\text{-}Vorpommern, Schleswig\text{-}Holstein), \\
& borders(Brandenburg, Mecklenburg\text{-}Vorpommern), \\
& borders(Niedersachsen, Mecklenburg\text{-}Vorpommern), \\
& borders(Sachsen\text{-}Anhalt, Mecklenburg\text{-}Vorpommern), \\
& borders(Schleswig\text{-}Holstein, Mecklenburg\text{-}Vorpommern), \\
& location(Rostock, Mecklenburg\text{-}Vorpommern), \\
& location(Schwerin, Mecklenburg\text{-}Vorpommern), \\
& capital\_of(Schwerin, Mecklenburg\text{-}Vorpommern)
\end{aligned}
$$

Now, computing the LGG of these two clauses yields the clause (compare Sec. 3.2 for the full clause):

$$
\begin{aligned}
answer(X) \leftarrow\ & state(X), location(X), inhabitants(X, Y), \\
& borders(X, V_{1,1}), ..., borders(X, V_{3,4}), \textit{(12 border-predicates)} \\
& borders(V_{1,1}, X), ..., borders(V_{3,4}, X), \textit{(12 inverse border-predicates)} \\
& borders(V_{*,5}, V_{1,*}), ..., borders(V_{6,*}, V_{*,4}), \textit{(24 border-predicates)} \\
& location(P, X), location(Q, X), capital\_of(Q, X)
\end{aligned}
$$

This clause can then be reduced to the following by removing redundant literals:

$$answer(X) \leftarrow state(X), location(X), inhabitants(X,Y), borders(X,Z), borders(Z,X), location(P,X), capital\_of(Q,X)$$

This clause covers 15 of the 16 original answers, but we can apply the procedure to remove irrelevant literals by considering one literal at a time according to our order (see Section 3.4), such that the following literals are removed in the indicated order as their removal does not increase the extension:

1. location(P,X) (covering 15 examples)
2. capital_of(Q,X) (covering 15 examples)
3. inhabitants(X,Y) (covering 15 examples)
4. borders(X,Z) (covering 15 examples)
5. borders(Z,X) (covering in addition Bremen)
6. location(X) (covering 16 examples)

This produces the clause: $answer(X) \leftarrow state(X)$ for our example. This clause then covers exactly the original answers and no others. The key here to cover also the example *Bremen* is thus the removal of the two `borders` predicates.

While this is not part of our current implementation of the system, an appropriate natural language answer could be generated from this clause, thus having *"All states (have a capital)"* as final intensional answer.

Thus, reducing the clause after each step has the effect that a correct clause can be derived with one LGG computation (one pass through the algorithm) for our example. In case the clause is not reduced after each step (by removing irrelevant literals), the algorithm needs to make 7 iterations, considering the examples: *Mecklenburg-Vorpommern, Rheinland-Pfalz, Hamburg, Schleswig-Holstein, Thüringen, Sachsen-Anhalt* and *Bremen*. Due to space limitations, we do not describe this procedure in more detail.

3.6. A note on complexity and scalability

Concerning the worst-case complexity of our approach consider the algorithm in Fig. 4 and let n be the number of answers. Each of the maximally n loops of the algorithm involves: i) a query to the knowledge base to construct a clause (line 6), ii) the computation of an LGG (line 7), iii) the evaluation

of a query with respect to the knowledge base (line 8) and iv) the reduction of the clause (line 13).

The main bottleneck is the computation of the LGG c (line 7), the size of which can get exponential in the number of the input clauses. It is bound by $O(m^n)$ where n is the number of answer items in the extensional answer and m is the maximum size of a specific clause for an entity in the knowledge base (roughly the maximum number of other entities an entity can be directly related to). This increase of c also affects the evaluation of the query with respect to the knowledge base (line 8).

The reduction of the clause (line 13) is again linear in the number of atoms of the clause to be reduced, which can be exponential as mentioned above.

While the algorithm does not seem very tractable at first sight, it turns actually out that it is very scalable when considering standard scenarios: the complexity is primarily determined by the number n of answer items returned, which can be assumed not to be too high for meaningful questions which do not ask for the complete extension of some concept (e.g. asking for all cities in the world). Our approach never operates on the full knowledge base but only on the answers returned, such that it can in principle scale to arbitrary large datasets as long as the number of answers returned is reasonable. The second source of complexity is the maximum number m of entities an entity is connected to. On the one hand, this is determined by the schema of the dataset and all the relationships that are possible. Larger datasets are typically poor from a schema point of view (e.g. the DBLP dataset).

The performance of our algorithms is thus highly dependent on the connectedness of the entities in the knowledge base. If the connectedness ratio is moderate, then the algorithm can certainly scale to arbitrarily big datasets.

4. Evaluation

The empirical evaluation of the system has been carried out with respect to a dataset used previously for the evaluation of the ORAKEL natural language interface (see [2])[5]. We analyze in detail to what extent the intensional

[5]The dataset is available at `http://www.cimiano.de` → Projects (Tab) → Datasets and other material → ORAKEL

answers produced by the system are "useful" and also discuss how the intensional answers can help in detecting errors in the knowledge base. Thus, we first describe the used dataset in more detail, then we discuss the usefulness of the generated intensional answers. After that, we show how intensional answers can be used to detect flaws (incomplete knowledge) in the knowledge base and also present some observations with respect to the time performance of our algorithm.

4.1. Dataset

The dataset previously used for the evaluation of the ORAKEL natural language interface consists of 463 wh-questions about German geography. These questions correspond to real queries by users involved in the experimental evaluation of the ORAKEL system (see [2]). The currently available system is able to generate an F-Logic query for 245 of these, thus amounting to a recall of 53%. For 205 of these queries, OntoBroker delivers a non-empty set as answer. Obviously, it makes sense to evaluate our approach to the generation of intensional answers only on the set of queries which result in a non-empty extension, i.e. the 205 queries mentioned before. Of these, the ILP-based approach is able to generate a clause for 169 of the queries. Roughly, we are thus able to generate intensional answers for about 83% of the relevant questions in our dataset. Here, only those questions are regarded as relevant which are translated by ORAKEL into an appropriate F-Logic query and for which OntoBroker delivers a non-empty extension. In general, there are two cases in which our ILP-based approach is not able to find a clause (in 36 out of 205 cases):

- The concept underlying the answers is simply not learnable in the form of a (single) clause expressible using our language (about 67% of the cases). An example here is *'Where is a city with more than 1.000.000 inhabitants?'*, translated by ORAKEL into the query: $FORALL\ X \leftarrow EXISTS\ Y, Z\ X[location \rightarrow Y]\ Y : city\ AND\ Y[inhabitants \rightarrow Z]\ AND\ greater(Z, 1000000))$. In order to represent such a clause we would need to introduce numerical comparison operators into our clause language. However, note that it is not excluded that such answers can be represented with our clause language if there are other properties not requiring numerical comparisons which might be used to pick out the entities in question. Such an example is the question *'Which cities have more than 1500000 inhabitants?'* (Answers: Berlin, Hamburg),

for which our approach generates the answer *'All (the cities) that are located at the A24 and which a river flows through'*. Another example is *"Which rivers flow through a capital?"*, which is translated into the query $FORALL\ X \leftarrow EXISTS\ Y, Z\ X[flows\_through \rightarrow Y]\ AND\ Y[capital\_of \rightarrow Z]$. No intensional clause is produced in this case as it requires to explore the further neighborhood of the entity Y to add the essential literal $capital\_of(Y, Z)$

- The answer to the query is the result of counting (about 33% of the cases), e.g. *"How many cities are in Baden-Württemberg?"*. Such numeric constants are not first-order citizens in the knowledge base (but actually elements of a certain datatype) and have no properties one could use to describe them.

Given this dataset, we first proceeded to analyze the usefulness of the intensional answers. For this purpose, we examined all the answers and determined whether the user learns something new from the answer or it merely constitutes a rephrasing of the question itself.

4.2. Analysis of Intensional Answers

For the 169 queries for which our approach is able to generate an intensional answer, it turned out that in 140 cases, the intensional answer could be actually regarded as *"useful"*, i.e. as not merely rephrasing the question and thus giving new information to the user. For example, to the question: *"Which rivers do you know?"*, the intensional answer *"All rivers"* does not give any additional knowledge, but merely rephrases the question. Of the above mentioned 140 useful answers, 97 are intensional answers describing additional properties of a single answer. Thus, for only 43 of the 140 useful answers the bottom-up generalization algorithm was actually used.

In order to further analyze the properties of intensional answers, we determined a set of 54 questions which, given our observations above, could be suitable for presenting intensional answers. For the new 54 queries, we received as results 49 useful intensional answers and 1 answer considered as not useful. For the other 4 questions, no consistent clause could be found. Together with the above mentioned 140 queries the overall number of useful intensional answers for the actual knowledge base is 189. The queries which provided useful answers can be classified into the four three types:

- Questions asking for all instances with a special property which turn out to coincide exactly with the extension of some atomic concept in the ontology (about 28% of the cases). The user learns from such an answer that all the instances of the atomic class in question share the property he/she is querying for. An example is the question: *"Which states have a capital?"*, which produces the answer $answer(X) \leftarrow state(X)$ or in natural language simply *"All states*".

- Questions asking for the relation with a specific entity which is paraphrased (possibly because the user doesn't know or remember it) (16% of the cases) Examples of such questions are: *"Which cities are in a state that borders Austria?"* or *"Which capitals are passed by a highway which passes Flensburg?"*. In the first case, the state in question which borders Austria is *"Bayern"* and the intensional answer is correctly: *"All the cities which are located in Bayern.*" In the second case, the highway which passes Flensburg is the A7, so the intensional answer is *"All the capitals located at the A7"*.

- Questions about properties of entities which lead to the description of additional properties shared by the answer set as intensional answer (about 5% of the cases). For example, the question *"Which states do three rivers flow through?"* produces the intensional answer *"All the states which border Bayern and which border Rheinland-Pfalz"*, i.e. *Hessen and Baden-Württemberg.* Another example is the question: *"Which rivers flow through more than 5 cities?"* (Answer: Rhein and Ruhr), yielding the intensional answer: *"All the rivers which flow through Duisburg"*. A further interesting example is the question *"Which cities are bigger than München?"* with the intensional answer: *"All the cities located at highway A24 and which a river flows through"*. These are the cities of Berlin and Hamburg. Such answers provide additional knowledge to the user about the entities in the extension, i.e. learning that Berlin and Hamburg are the only cities bigger than Munich, but also the only ones which are located at highway A24 and at some river.

- Questions which yield a single answer as a result and for which the intensional answer describes additional properties (51%). An example would be: *"What is the capital of Baden Württemberg?"*, where the extensional answer would be *"Stuttgart"* and the intensional answer (paraphrased in natural language) *"Stuttgart is located at the highways*

A8, A81, A85 and A831. It has 565.000 inhabitants and is the capital of Baden Württemberg. The Neckar flows through Stuttgart."

4.3. Debugging the Knowledge Base

In addition to providing further information about the answer space, a key benefit of intensional answers is that they allow to detect errors in the knowledge base. We found that in some cases the intensional answer produced was not the expected one, hinting at possible modeling flaws in the knowledge base in question. We give a few examples to illustrate this idea:

- For example, for the question *"Which cities have more than 9 inhabitants?"*, we would expect the intensional answer *"All cities"*. Yet, we obtained the answer $answer(X) \leftarrow city(X) \wedge inhabitants(X, Y)$, i.e. *"all the cities with a number of inhabitants"*. This hints at the fact that the number of inhabitants is not defined for all declared cities in the knowledge base. A closer inspection of the knowledge base revealed that some Swiss cities are mentioned in the knowledge base because some German rivers have their origin in Switzerland, but no number of inhabitants is specified for these cities.

- As another example, the query *"Which river has an origin?"* yielded as intension: $answer(X) \leftarrow river(X) \wedge length(X, Y)$ (*"All rivers which have a length"*) instead of *"All rivers"*. This is due to the fact that there is one instance of river for which neither a length nor an origin is specified. This instance is the river *Isar*.

This shows that intensional answers can be useful to detect where information in the knowledge base is missing.

4.4. Query-Driven Knowledge Base Refinement

Another application of intensional answers is knowledge base refinement. Consider our initial example, where we were looking for states having a capital, formally we queried for all x with $state(x)$ as well as $capital\_of(y, x)$ for some y or, more formally, for all x satisfying $state(x) \wedge \exists y(capital\_of(y, x))$. The intensional answer we obtained from the system tells us that these are exactly those x satisfying just $state(x)$. Therefore, the logical proposition $\forall x(state(x) \wedge \exists y(capital\_of(y, x)) \leftrightarrow state(x))$ holds with respect to all individuals recorded in our knowledge base. While the "$\rightarrow$" direction of this

Reduction	**Time (in sec.)**			**# Iterations**		
	∅	min	max	∅	min	max
after each LGG computation	0.528	0.08	13.61	0.621	0	5
at the end	0.652	0.07	33.027	1.702	0	73

Table 1: Performance measurements for our testsuite

proposition is trivially true, the "←" direction provides information about the domain telling that every state has a capital. Now, there are two possibilities: This might be a proposition that is just incidentally true for all the states recorded in the knowledge base and that could be disproven by adding an (existing but not yet recorded) state that has no capital to the knowledge base. The other case would be that having a capital is a necessary feature for every possible state such that the above proposition has to be universally valid. In this case, the proposition $(state(x) \rightarrow state(x) \wedge \exists y(capital\_of(y, x))$ or its equivalent in the used formalism) can be added to the ontology as a logical axiom, provided the knowledge modeling language used offers the required expressivity for doing so.

This strategy provides a way of continuously refining the knowledge stored in the system by the user. If a query imposed by the user returns a nontrivial intensional answer, the user can be asked, whether the original query implies the delivered intensional answer and/or the other way around. If he denies, he might be able to input "witness individuals" for this denial, otherwise the confirmed propositions are added to the knowledge base. Either way, the information system is enriched by new knowledge that might be useful in further queries.

4.5. Performance Analysis

Finally, we have also carried out a performance analysis of the component for inducing intensional answers. Table 1 shows the average time and iterations needed to compute an intensional answer. In particular, we tested two configurations of our system: one corresponding to a version in which the clause is reduced after each LGG computation and one in which the clause is only reduced at the end. A first analysis of the time performance reveals that our approach can be used efficiently, taking on average about half a second to compute an intensional answer. As we see extensional and intensional answers as equally important and assume that both are shown to the user, we can even assume that first the extensional answer is computed and shown

already to the user while the intensional answer is still computed. While the results differ considerably with respect to the maximum for the different configurations of the system, it is interesting to see that they hardly differ in the average case. In fact, when the reduction is performed after each LGG computation, the average time is only very slightly under the average time for the version in which the clause is reduced only at the end. Nevertheless, the reduction clearly has an influence on the number of iterations needed, which is much lower than in the case where the reduction is only applied at the end.

5. Related Work and Discussion

We have presented an approach for computing intensional answers to a query formulated in natural language with respect to a given knowledge base. The approach relies on Inductive Logic Programming techniques, in particular bottom-up clause generalization, to compute a clause subsuming the extensional answer as returned by the inference engine, which evaluates queries to the knowledge base. This clause represents the intensional answer to the question and could be transformed back into natural language. The latter step is not part of our present contribution and thus remains for future work. The idea of delivering intensional answers to queries is relatively straightforward and has been researched in the context of knowledge discovery from databases (see e.g. [15], [16], [17]), but also logic programs (see e.g. [18]).

Intensional query answering (IQA) emerged as a subject in the research of cooperative response generation, which has been a topic of some works related to natural language interfaces (compare [19] and [20]). The work closest to ours is the one of Benamara [20] who also presents an approach to induce intensional answers given the extension of the answer. Instead of using ILP, Benamara relies on a rather ad-hoc approach in which parts of the answer are replaced by some generalized description on the basis of the given knowledge base. This generalization is guided by a *"variable depth intensional calculus"* which relies on a metric measuring the distance between two concepts as the number of arcs and the inverse proportion of shared properties. On the one hand, the mechanism for generating intensional answers seems quite adhoc in comparison to ours, but, on the other hand, the WEBCOOP system of Benamara is able to generate natural language answers using a template-based approach.

Our approach is very related to research in the area of generating referring expressions (compare [21], [22], [23], [24], [25]). The problem is formulated in a way that resembles our problem formulation in [21], where the task is defined as follows: Given a set of entities S (the *target set*) to be described, the task is to find a *distinguishing description* that is true of all elements in S but of no other entity in a set of *potential distractors* $D := C \backslash S$, where C is the *context set*. In research on generating referring expressions (typically generating noun phrases and definite descriptions in particular) the context set is assumed to be what is accessible to the reader or hearer in the situation in question. In our settings, the target set S consists of the answers to the original query, the context set C are all those entities available in the knowledge base (as closed world) and $D := C \backslash S$ is the set of negative examples. The tasks are thus conceptually similar and consist in finding a *distinguishing description* which exactly picks out the elements of the target set from the larger context. Our solution to this problem has been a bottom-up generalization algorithm based on ILP, which starts from a clause representing all the factual information gathered for the entity in question from the knowledge base. Answers are added iteratively one at a time to yield a most specific generalized description until a description which uniquely picks out the target set is found. An important characteristic of our approach is that we only search for single clauses and stop when the first consistent clause is found, which can then be reduced by removing certain literals. Our algorithm is thus greedy and lacks any completeness properties which would be needed to find the clause which is maximal or minimal with respect to some criteria (e.g. minimizing the length of the description [25]). Clearly, the problem of minimizing or maximizing a certain criterion requires enumerating all solutions and is NP-hard (see [22]). Gardent deals with this problem by a formulation of the problem as a constraint satisfaction problem and using efficient constraint programming techniques for this purpose (see [25]). While our approach is certainly far from being complete, we have tried to approximate some criteria via some heuristics. For example, the order by which we remove irrelevant literal aims at producing more compact and thus hopefully more intuitive clauses.

In the area of generating referring expressions, Dale and Reiter have presented a top-down (starting from the empty description) and incremental (in the sense that it does never back-track) algorithm which adds the best attribute at a time in a greedy fashion and stops when a distinguishing (consistent in our terms) description has been found (see [21]). In its original

fashion the incremental algorithm of Reiter and Dale only generates a description of a single referent by conjoining properties to yield a distinguishing description. Van Deemter [22] has convincingly shown that the incremental algorithm of Reiter and Dale is not complete in the sense that it can not be guaranteed that if there is an actual distinguishing description, it will also be found. Van Deemter then shows that under certain assumptions the algorithm can be extended to be complete. Further, he presents an extension to the algorithm dealing with sets of referents, one version dealing with sets in a trivial manner and a further version in which sets are 'first-order citizens' (elements of the power set of entities), thus allowing to define properties on whole sets rather than on elements of sets. Such descriptions are for example: *Sets of rivers which have the same origin*. This is clearly a property predicated of sets rather than of single elements. Horacek proposes a best-first (also top-down) search algorithm for generating referring expressions which scores the (partial) solutions found so far, adding in each loop new attributes to the best solution, scoring it and checking if it is better than the solutions found so far. This best-first search algorithm is extended in [24] to include constraints on surface form with the goal of producing more natural descriptions from a linguistic point of view as well as extending the description language by allowing to express exclusion (in addition to conjunction, negation and disjunction) explicitly. Dale and Haddock [26] present an extension to the incremental algorithm to handle relational properties. This is a challenge in a generation context as one would like to avoid recursive descriptions, which could emerge when describing an entity through its relation to another entity for which a description needs to be recursively generated. As we are not concerned with a generation setting here, we have no need to describe the entities standing in a relation to an entity in the answer further, such that we are not concerned with this problem. In this sense, while our approach certainly takes into account relational information, from one point of view it does so in a rather propositional manner as the entities to which the entities in the answer are related to are not further considered. However, from another point of view, the concrete values of a relation can be abstracted over in the induction process by inserting a variable, so that we are not really treating these relations only in a propositional manner. In this sense we also introduce new properties which hold for whole sets (i.e. the extension of a clause) in the line of Van Deemter and by suitable modifications of the generalization step (currently implemented via the LGG), we could also generate descriptions such as *All the rivers which have the same origin*,

which should be located somewhere between having a concrete constant at a given argument position and allowing any value (as indicated by a variable) in the generalization lattice. In contrast to the approaches of Horacek [23], [22] and [25] our approach is limited to conjunctive descriptions as the original incremental algorithm of Reiter and Dale [21].

As discussed above, there is a clear connection between the generation of intensional answers and the generation of referring expressions (GRE). The algorithms in the GRE literature loop over the set of descriptors adding descriptors in a top-down manner until the set of potential distractors is empty. This resembles the strategy adopted in other ILP approaches which perform in a top-down manner (e.g. FOIL [27]).

We have proceeded in a bottom-up way in our approach, adding one example at a time, generalizing the growing descriptions until it covers all positive examples but no negative ones.

Neither of these two strategies are specific to the problem nor technique adopted. Both in GRE and GIA (Generation of Intensional Answers), the search space of possible descriptions can be traversed in a bottom-up or top-down manner. Common to both problems is that finding all possible descriptions is NP-hard. Thus, more efficient, i.e. greedy, algorithms are needed that cut-down the search space while producing the most suitable, compact or economic solutions. Both in GRE and GIA we would certainly like to produce descriptions which users can grasp with minimal effort. Such preferences are encoded into algorithms in order to guide the search in the space of all possible descriptions, either in a greedy style as in our case and also in the case of the incremental algorithm of Reiter and Dale [21] or in a best-first fashion [23].

The dichotomy intension vs. extension is also prevalent in the mathematical area of Formal Concept Analysis (FCA, [28]) where lattice theory[6] is applied to model conceptual thinking. In terms of FCA, every formal concept comprises a set of objects (the extension) and a maximal set of attributes (the intension) that is shared by all objects and characterizes them. While original FCA deals with propositional logic only, the framework has been extended to description logics [29] and first order logics [30]. Building on the FCA algorithm of attribute exploration [31], techniques for interactive knowledge base refinement in those settings have been developed (cf. also

[6] more precisely: the theory of Galois connections

[32],[33], and [34]). Conceptually, those techniques are similar to the strategy presented in Section 4.4. However, all those methods aim at completely clarifying all logical interdependencies holding in a domain, leading to a large number of hypotheses subsequently presented to the user for decision. This has shown to be a time-consuming and tedious task. In contrast to that, the strategy outlined by us just comes up with one question to the user that is related to his current focus of interest anyway (as it is triggered by the according query posed by him).

6. Conclusion

While the idea of delivering intensional answers is certainly appealing and intuitive, and we have further shown that they can be computed in a reasonable amount of time, we have also argued that their benefit is not always obvious. While our approach is able to generate intensional answers for about 83% of the questions which can be translated by ORAKEL to a logical query and actually have an extension according to the inference engine, we have shown that in some cases the answer is actually θ-equivalent and thus also logically and extensionally equivalent to the question. In these cases, the intensional answer is not delivering additional information to the user. However, this case could be easily ruled out by checking θ-equivalence between the query and the clause returned as intensional answer. In these cases, the intensional answer can then be omitted. An interesting option to explore would be to determine the literals to be removed during the clause reduction on the basis of the given query to avoid producing an answer which is θ-equivalent, thus yielding more informative answers.

In the remaining cases, the intensional answers seem to indeed be delivering additional information about the knowledge base to a user (in some cases in a quite oracle-like fashion arguably). Thus, while we have considered such answers as "useful", a real proof of "usefulness" can only be provided in the form of user experiments, which we have not carried out.

While our work on generation of intensional answers (GIA) has been carried out in the context of the ORAKEL system, the approach presented is not specific to the latter in any way. The approach operates on the basis of the extensional answer returned by some system and merely requires a query interface to the knowledge base. In this sense the approach presented is generic and can be integrated into any natural language interface or question answering system relying on a knowledge base.

The most obvious avenues for future work are the following. First, the bottom-up rule induction approach should explore "larger environments" of the answer items for clause construction by not only considering properties of the answer item itself (i.e. related through one step in the KB) but further properties of the entities to which it is related. In the extreme case, all facts of the knowledge base could be selected as body of the constructed clause. Overall, increasing the environment considered to construct our clauses will also clearly affect the complexity of our approach (see Section 3.6). The additional gain in expressivity needs thus to be balanced with the increased complexity, a non-trivial issue.

Second, instead of only computing one single clause, we should also consider computing (disjunctive) sets of conjunctive descriptions, thus hopefully increasing the coverage of the approach. However, it could turn out that the disjunctions of intensional descriptions are much harder to grasp by users. The possibility of expressing disjunction might in some cases shorten the process of finding an intensional description. Using disjunction, a corresponding intensional description for our states example might be yielded in one step as *"All states that have a capital and border at least one state, as well as Bremen"*. While the increase in expressiveness in the language used allows to induce a description in one step, it is questionable if such an answer is perceived as more intuitive by users.

Further, the addition of negation and exclusion operators to our language should be investigated together with its implications on the formulation of the bottom-up generalization and its performance. The addition of exclusion would have for instance produced a different answer for the example question *Which river has an origin?*, with the answer *'All rivers except for the Isar'*, which would have been certainly very informative and would also show the incompleteness of the knowledge base in a straightforward and intuitive manner.

It would also be interesting to compare the algorithms suggested to produce distinguishing descriptions in research on generating referential expressions on our task.

Clearly, our approach is restricted to "closed" domains and assumes that all relevant information is modeled in the form of a knowledge base from which the intensional answers are generated. We do not see any way how our approach could be straightforwardly extended to an open domain scenario. It is even questionable if extending the approach to handle unstructured and open domains is desirable at all. After all, the advantage of a

question answering approach operating on a closed domain modeled in terms of a knowledge base is that counting, aggregation, comparisons etc. can be performed, which is very hard, if not impossible, to achieve in unstructured and open-domain settings.

Finally, the task of generating natural language descriptions of the intensional answers represents an obvious continuation of our work.

On a more general note, we hope that our work contributes to stimulating discussion on a research issue which has not been on the foremost research frontier lately, but seems crucial towards achieving more natural interactions with information systems.

Acknowledgements:. This research has been funded by the Multipla (grant number 38457858) and ReaSem projects, both sponsored by the Deutsche Forschungsgemeinschaft (DFG).

References

[1] T. Strzalkowski, S. Harabagiu, Advances in Open Domain Question Answering, Vol. 32 of Text, Speech and Language Technology, Springer, 2006.

[2] P. Cimiano, P. Haase, J. Heizmann, M. Mantel, R. Studer, Towards portable natural language interfaces to knowledge bases: The case of the ORAKEL system, Data and Knowledge Engineering (DKE) 62 (2) (2007) 325–354.

[3] I. Androutsopoulos, G. Ritchie, P. Thanisch, Natural language interfaces to databases–an introduction, Journal of Language Engineering 1 (1) (1995) 29–81.

[4] A. Copestake, K. S. Jones, Natural language interfaces to databases, Knowledge Engineering Review 5 (4) (1989) 225–249, special Issue on the Applications of Natural Language Processing Techniques.

[5] R. Muskens, Talking about trees and truth-conditions, Journal of Logic, Language and Information 10 (4) (2001) 417–455.

[6] A. Joshi, Y. Schabes, Tree-adjoining grammars, in: Handbook of Formal Languages, Vol. 3, Springer, 1997, pp. 69–124.

[7] P. Cimiano, P. Haase, J. Heizmann, M. Mantel, Orakel: A portable natural language interface to knowledge bases, Tech. rep., Institut AIFB, Universität Karlsruhe (March 2007).

[8] P. Blackburn, J. Bos, Representation and Inference for Natural Language - A First Course in Computational Semantics, CSLI Publications, 2005.

[9] M. Kifer, G. Lausen, J. Wu, Logical foundations of object-oriented and frame-based languages, Journal of the ACM 42 (1995) 741–843.

[10] S. Decker, M. Erdmann, D. Fensel, R. Studer, Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information, in: Database Semantics: Semantic Issues in Multimedia Systems, Kluwer, 1999, pp. 351–369.

[11] D. McGuiness, F. van Harmelen, OWL Web Ontology Language Overview, W3C Recommendation, available at http://www.w3.org/TR/owl-features/ (February 2004).

[12] N. Guarino, P. Giaretta, Ontologies and knowledge bases: Towards a terminological clarification, in: N. Mars (Ed.), Towards Very Large Knowledge Bases, IOS Press, 1995.

[13] N. Lavrac, S. Dzeroski, Inductive Logic Programming: Techniques and Applications, Ellis Horwood, 1994.

[14] T. Mitchell, Machine Learning, McGraw-Hill, 1997.

[15] A. Motro, Intensional answers to database queries, IEEE Transactions on Knowledge and Data Engineering 6 (3) (1994) 444–454.

[16] S.-C. Yoon, I.-Y. Song, E. K. Park, Intelligent query answering in deductive and object-oriented databases, in: Proceedings of the 3rd International Conference on Information and knowledge management (CIKM), 1994, pp. 244–251.

[17] P. A. Flach, From extensional to intensional knowledge: Inductive logic programming techniques and their application to deductive databases, in: Transactions and Change in Logic Databases, Vol. 1472, Springer-Verlag, 1998, pp. 356–387.

[18] G. D. Giacomo, Intensional query answering by partial evaluation, Journal of Intelligent Information Systems 7 (3) (1996) 205–233.

[19] T. Gaasterland, P. Godfrey, J. Minker, An overview of cooperative answering, Journal of Intelligent Information Systems 1 (2) (1992) 123–157.

[20] F. Benamara, Generating intensional answers in intelligent question answering systems, in: Proceedings of the International Conference on Natural Language Generation (INLG), Vol. 3123, Springer Verlag, 2004, pp. 11–20.

[21] E. Reiter, R. Dale, Generating definite NP referring expressions, in: Proceedings of the 14th International Conference on Computational Linguistics (COLING), 1992.

[22] K. van Deemter, Generating referring expressions: Boolean extensions of the incremental algorithm, Computational Linguistics 28 (1) (2002) 37–52.

[23] H. Horacek, A best-first search algorithm for generating referring expressions, in: Proceedings of the 10th European Chapter of the Association for Computational Linguistics (EACL'03), 2003, pp. 103–106, short paper.

[24] H. Horacek, On referring to sets of objects naturally, in: Proceedings of the 3rd International Conference on Natural Language Generation (INLG), 2004, pp. 70–79.

[25] C. Gardent, Generating minimal definite descriptions, in: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics and the 8th European Chapter for the Association for Computational Linguistics (ACL-EACL), 1997, pp. 206–213.

[26] R. Dale, N. Haddock, Generating referring expressions involving relations, in: Proceedings of the 5th Conference on the European Chapter of the Association for Computational Linguistics (EACL), 1991, pp. 161–166.

[27] J. Quinlan, Learning logical definitions from relations, Machine Learning 5 (1990) 239–266.

[28] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[29] S. Rudolph, Relational exploration - combining description logics and formal concept analysis for knowledge specification, Ph.D. thesis, Technische Universität Dresden, Germany (2006).

[30] M. Zickwolff, Rule exploration: First order logic in formal concept analysis, Ph.D. thesis, FB4, TH Darmstadt (1991).

[31] B. Ganter, Two basic algorithms in concept analysis, Tech. Rep. 831, FB4, TH Darmstadt (1984).

[32] F. Baader, B. Ganter, B. Sertkaya, U. Sattler, Completing description logic knowledge bases using formal concept analysis, in: M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 230–235.

[33] J. Völker, S. Rudolph, Lexico-logical acquisition of OWL DL axioms, in: R. Medina, S. A. Obiedkov (Eds.), Proceedings of the International Conference on Formal Concept Analysis (FCA), Vol. 4933 of Lecture Notes in Computer Science, Springer, 2008, pp. 62–77.

[34] J. Völker, S. Rudolph, Fostering web intelligence by semi-automatic OWL ontology refinement, in: Proceedings of the International Conference on Web Intelligence (WI), 2008, pp. 454–460.