Intensional Question Answering using ILP: What does an answer mean?

Philipp Cimiano, Helena Hartfiel, Sebastian Rudolph

Institute AIFB, Universität Karlsruhe (TH)

Abstract. We present an approach for computing intensional answers given a set of extensional answers returned as a result of a user query to an information system. Intensional answers are considered as descriptions of the actual answers in terms of properties they share and which can enhance a user's understanding of the answer itself but also of the underlying knowledge base. 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 its potential for helping to detect flaws in the knowledge base is discussed. While the approach is used in the context of a natural language question answering system in our setting, it clearly has applications beyond.

1 Introduction

Question answering systems for unstructured ([1]) or structured ([2]) information have been a focus of research since a few decades now. They are a crucial component towards providing intuitive access for users to the vast amount of information available world wide offered by information sources as heterogeneous as web sites, 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 with an extensional answer here is a list of those facts which fulfill the query. 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 not maximally informative.

A 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)."*, thus informing a user about characteristics that all the elements in the answer set share. We will refer to such descriptions which go beyond mere enumeration of the answers as *intensional*

answers (IAs). The intension is thus a description of the elements in the answer set in terms of features common to all of them. 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.

We could certainly argue that this is over-answering the question. However, in many cases a listing of facts does not help the user to actually understand the answer. First, the extension might be simply too large for a user to make sense of it. In this case, a compact representation of the answer in terms of an intensional description might be useful. In other cases, the user can be dissatisfied with the answer because he simply doesn't know why the elements in the extension actually answer the query. In the above example, the user doesn't learn from the extensional answer that all German federal states have a capital if he doesn't know that the answer consists exactly of all the federal states in Germany. Finally, we will see in the remainder of this paper how intensional answers can enhance the understanding about relations in the knowledge base and can also help in discovering modeling errors.

In this paper, we thus present an approach for computing "intensions" of queries given their extensions and a particular 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.We discuss the approach in more detail in Section 2. In Section 3, 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 and conclude in Section 4.

2 Approach

2.1 Architecture

The aim of the work described in this paper has been to extend the ORAKEL natural language interface with capabilities for delivering intensional answers. ORAKEL is a natural language interface which translates wh-questions into logical queries and evaluates them with respect to a given knowledge base. ORAKEL implements a compositional semantics approach in order to compute the meaning of a wh-question in terms of a logical query. It is able to deal with arbitrary knowledge representation languages and paradigms (see [2]). However in this work, the implementation is based on F-Logic [3] and Ontobroker [4] is used as the underlying inference engine. For more details about the ORAKEL system, the interested reader is referred to [2]. The workflow of the system is depicted in Figure 1. When the user asks a natural language question referring to 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. While the extensional answers are displayed to the user, they are

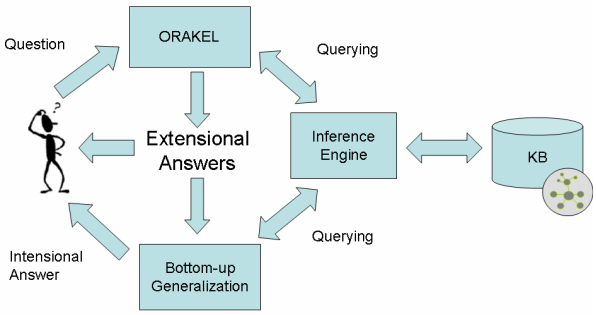

Fig. 1. Workflow of the system

also forwarded to the component which generates intensional answers (named *Bottom-up Generalization* in the figure). This component provides a hypothesis (in the form of a clause) based on the extensional answers by means of an ILP-algorithm. This hypothesis is the intensional answer and is displayed in addition to the extensional answers. 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 issue for future work. We assume that even in the case of a useful hypothesis, the user is interested in the direct answers as well, such that we do not forego their presentation. The displaying of the direct answers goes 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.

In this article we present an approach for computing *intensions* of queries given their extensions and a particular knowledge base. Our approach builds on Inductive Logic Programming (ILP) [5] as an inductive learning approach to learn descriptions of examples (our answers) in the form of clauses. In particular, it is based on bottom-up clause generalization, which computes a clause subsuming the extension of an answer and thus representing an "intensional" answer in the sense that it describes it in terms of features which all elements in the extension share. In particular, the system iteratively calculates least general generalizations (LGGs) ([5], pp. 36) for the answers by adding one answer at a time and generalizing the clause computed so far. According to our 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 program clauses. This search space of program 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 [5], pp. 33-37).

We make sure that the resulting clause is consistent by evaluating it with respect to the knowledge base, relying on the inference engine in order to check that it does not cover other examples than the extensional answers. In fact,

```
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 c is inconsistent (i.e.Answers ∩ Answers' ⊂ Answers')
10        then
11              return ∅ ( no consistent clause can be found )
12
13      c'' = reduceClause(c, KB, Answers)
14      if c'' covers all answers
15        then
16              return c'';
17
18  return reduceClause(c, KB, Answers);
```

Fig. 2. Generalization Algorithm (calculating a reduced clause after each LGG computation)

the resulting clause can be straightforwardly translated into a corresponding query to the knowledge base. Our learning algorithm is thus similar to the FindS algorithm described in [6] 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 in the sense that it does not cover negative examples, then an appropriate clause has been found. Otherwise, there is no clause which covers the positive examples without overgeneralizing. We explain this procedure more formally in the next section.

2.2 Generalization Algorithm

The generalization algorithm is given in Figure 2. The algorithm takes as input the set of (extensional) answers (*Answers*) as well as the knowledge base. The aim is to generate a hypothesis which exactly covers the extensional answers. A hypothesis in our approach is actually a clause of literals that constitutes a non-recursive definite program clause. For the hypothesis generation, the ILP-learner first constructs a specialized clause based on the first positive example. As head of this clause we use an artificial predicate *answer*. This is done by the `constructClause` procedure, which returns a clause for a certain individual in the knowledge base consisting of all the factual information that can be retrieved for it from the knowledge base. If there is only one answer, then the constructed clause is exactly the intensional answer we are searching for. In such a case, the

user thus yields 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, we loop over these and compute the LGG of the clause c constructed so far and the clause c' constructed on the basis of the next answer a'. The resulting clause is sent as query to the inference engine, which returns the extension of the clause c on the basis of the given knowledge base. If the clause is inconsistent, 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 $\emptyset$. 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 and we test if this reduced clause covers exactly the original answers. If this is the case we return the reduced clause c'' and are done. 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'. This algorithm returns a clause in case it covers exactly the set of extensional answers. The LGG is essentially computed as described in [5], pp. 40 with the only exception that we do not allow functions as terms.

Clause Reduction The following algorithm performs the reduction of the clause:

```
reduceClause( Clause c, KnowledgeBase KB, Set Answers)
 1  List literals = orderLiterals(c);
 2  for i = 1 to |literals|
 3      do
 4          c' = remove(c, l_i);
 5          Answers' = evaluateQuery(query(c'), KB);
 6          if c' is consistent with respect to Answers
 7            then
 8                  c = c'
 9
10  return c;
```

where `evaluateQuery` essentially evaluates a clause formulated as query with respect to the existing knowledge base; `orderLiterals` orders the literals in a clause by an order that fulfills:

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

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 2) or only at the end. Reduction at the end is unproblematic as the clause covers exactly 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. It is important to mention that the reduction after each iteration is not strictly necessary as it finally yields the same intensional answer. However, our experiments show that by applying the reduction procedure after each iteration we can reduce the number of iterations and speed up our algorithm in most cases.

2.3 A Worked Example

Let's consider our running example, the question: *"Which states have a capital?"*. ORAKEL translates this question into the following logical query: $FORALL\ X \leftarrow EXISTS\ Y\ X : state \wedge X[capital \rightarrow Y].\ orderedby\ X$. 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("Thueringen")
answer("Sachsen-Anhalt")	answer("Sachsen")
answer("Bremen (Bundesland)")	answer("Niedersachsen")
answer("Brandenburg (Bundesland)")	answer("Berlin (Bundesland)")
answer("Baden-Wuerttemberg")	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. 2.

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:

answer("Saarland") ← state("Saarland"), location("Saarland"),
inhabitants("Saarland",1062754.0), borders("Saarland","Rheinland-Pfalz"),
borders("Saarland","France"), borders("Saarland","Luxembourg"),
borders("Rheinland-Pfalz","Saarland"), location("Saarbruecken","Saarland"),
capital_of("Saarbruecken","Saarland"), borders("France","Saarland"),
borders("Luxembourg","Saarland"), flows_through("Saar","Saarland")

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

answer("Mecklenburg-Vorpommern") ← state("Mecklenburg-Vorpommern"),

location("Mecklenburg-Vorpommern"),
inhabitants("Mecklenburg-Vorpommern",1735000.0),
borders("Mecklenburg-Vorpommern","Brandenburg"),
borders("Mecklenburg-Vorpommern","Niedersachsen"),
borders("Mecklenburg-Vorpommern","Sachsen-Anhalt"),
borders("Mecklenburg-Vorpommern","Schlesw.-Holstein"),
borders("Brandenburg","Mecklenburg-Vorpommern"),
borders("Niedersachsen","Mecklenburg-Vorpommern"),
borders("Sachsen-Anhalt","Mecklenburg-Vorpommern"),
borders("Schlesw.-Holstein","Mecklenburg-Vorpommern"),
location("Rostock","Mecklenburg-Vorpommern"),
location("Schwerin","Mecklenburg-Vorpommern"),
capital_of("Schwerin","Mecklenburg-Vorpommern")

Now, computing the LGG of these two clauses yields the clause:

answer(X) ← state(X), location(X), inhabitants(X,Y),
borders(X,Z),..., borders(X,O), *(12 redundant border(X,_)-predicates)*
borders(Z,X),..., borders(O,X), *(12 redundant border(_,X)-predicates)*
location(P,X), location(Q,X), capital_of(Q,X)

This clause can then be reduced to the following by removing redundant literals:[1]

answer(X) ← state(X), location(X), inhabitants(X,Y), borders(X,Z),
borders(Z,X), location(P,X), capital_of(Q,X)

Finally, irrelevant literals can be removed, resulting in a clause which does not increase the number of negative examples, but possibly increases the number of positives ones, thus resulting in the clause $answer(X) \leftarrow state(X)$ for our example. This clause then covers exactly the original answers and no others. 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 in one step 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 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]). 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. Thus, we first describe the

[1] Note that this is another kind of reduction than the one presented in Section 2.2. As opposed to the "empirical" reduction described there, the removal of redundant literals yields a clause which is logically (and not just extensionally) equivalent to the original one.

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 statements) in the knowledge base and also present some observations with respect to the time performance of our algorithm.

3.1 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 with one single clause (about 67% of the cases).
- The answer to the query is the result of counting (about 33% of the cases), e.g. *"How many cities are in Baden-Württemberg?"*.

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.

3.2 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 following 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 class in question share the property he 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?"*, 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.
- Questions which yield a single answer as a result and for which the intensional answer describes additional properties. 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 and A85 and A831. Stuttgart has 565.000 inhabitants and is the capital of Baden Württemberg. The Neckar flows through Stuttgart."*

3.3 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, which hints at the fact that something is wrong with the knowledge base in question. We give a few examples to illustrate this idea:

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

- For example, for the question *"Which cities have more than 9 inhabitants?"*, we would expect the intensional answer *"All cities"*. However, our approach yields the answer $answer(x) \leftarrow \exists y\ city(x) \wedge inhabitants(x, y)$, i.e. *"all the cities for which the number of inhabitants is specified"*. 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.
- The question *"Which cities are located in a state?"* yields no intensional answer at all, while we would expect the answer "all cities". However, as mentioned above, there are cities in Switzerland for which a corresponding state is missing.
- The query *"Which river has an origin?"* yields as intension: $answer(x) \leftarrow \exists y\ river(x) \wedge length(x, y)$ (*"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 the information in the knowledge base is not complete.

3.4 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.

4 Related Work and Conclusion

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. [7], [8], [9]), but also logic programs (see e.g. [10]).

Intensional query answering (IQA) emerged as a subject in the research of cooperative response generation, which has been a topic in many works related to natural language interfaces (compare [11] and [12]). The approach closest to ours is the one of Benamara [12] 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.

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. In the remaining cases, the intensional answers seem to indeed be delivering additional and useful information about the knowledge base to a user (in some cases in a quite oracle-like fashion arguably). The most obvious avenues for future work are the following. First, the bottom-up rule induction approach should explore "larger environments" of the (extensional) answers for clause construction by not only considering properties of the entity in the answer set but further properties of the entities to which it is related. 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. Finally, the task of generating natural language descriptions of the intensional answers seems the most pressing issue. On a more general note, we hope that our paper 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 and ReaSem projects, both sponsored by the Deutsche Forschungsgemeinschaft (DFG).

References

1. Strzalkowski, T., Harabagiu, S.: Advances in Open Domain Question Answering. Volume 32 of Text, Speech and Language Technology. Springer (2006)
2. Cimiano, P., Haase, P., Heizmann, J., Mantel, M., Studer, R.: 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. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995) 741–843
4. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In: Database Semantics: Semantic Issues in Multimedia Systems. Kluwer (1999) 351–369
5. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
6. Mitchell, T.: Machine Learning. McGraw-Hill (1997)
7. Motro, A.: Intensional answers to database queries. IEEE Transactions on Knowledge and Data Engineering **6**(3) (1994) 444–454
8. Yoon, S.C., Song, I.Y., Park, E.K.: Intelligent query answering in deductive and object-oriented databases. In: Proceedings of the third international conference on Information and knowledge management (CIKM). (1994) 244–251
9. Flach, P.A.: From extensional to intensional knowledge: Inductive logic programming techniques and their application to deductive databases. In: Transactions and Change in Logic Databases. Volume 1472. Springer-Verlag (1998) 356–387
10. Giacomo, G.D.: Intensional query answering by partial evaluation. Journal of Intelligent Information Systems **7**(3) (1996) 205–233
11. Gaasterland, T., Godfrey, P., Minker, J.: An overview of cooperative answering. Journal of Intelligent Information Systems **1**(2) (1992) 123–157
12. Benamara, F.: Generating intensional answers in intelligent question answering systems. In: Natural Language Generation. Volume 3123. Springer Verlag Berlin Heidelberg (2004) 11–20