Approximate Instance Retrieval on Ontologies*

Tuvshintur Tserendorj[1], Stephan Grimm[1], and Pascal Hitzler[2]

[1] FZI Research Center for Information Technology, Karlsruhe, Germany
[2] Kno.e.sis Center, Wright State University, Dayton, Ohio

Abstract. With the development of more expressive description logics (DLs) for the Web Ontology Language OWL the question arises how we can properly deal with the high computational complexity for efficient reasoning. In application cases that require scalable reasoning with expressive ontologies, non-standard reasoning solutions such as approximate reasoning are necessary to tackle the intractability of reasoning in expressive DLs. In this paper, we are concerned with the approximation of the reasoning task of instance retrieval on DL knowledge bases, trading correctness of retrieval results for gain of speed. We introduce our notion of an approximate concept extension and we provide implementations to compute an approximate answer for a concept query by a suitable mapping to efficient database operations. Furthermore, we report on experiments of our approach on instance retrieval with the Wine ontology and discuss first results in terms of error rate and speed-up.

1 Introduction

For description logics, there are two main approaches to reasoning. Tableaux-based methods [1] implemented in tools such as Pellet [2] and Racer [3] have been shown to be efficient for complex TBox reasoning tasks with expressive DLs. In contrast, the reasoning techniques based on reduction to disjunctive datalog as implemented in KAON2 [4] scale well for large ABoxes, with support for the DL $\mathcal{SHIQ}$. Besides these two directions, other approaches such as rule engines and database-based techniques scale very well for large ABoxes, but are in principle limited to lightweight language fragments [5].

Observing the application domain of these approaches, an issue which remains to be investigated is the problem of scalable reasoning over expressive ontologies with large ABoxes as well as complex or large TBoxes. From a theoretical point of view we know that it is impossible to find any tractable algorithm for reasoning over expressive ontologies due to the underlying high computational complexities [6]. Thus, non-standard reasoning solutions like approximate reasoning [7, 8] are helpful in time-critical applications when it is acceptable to sacrifice soundness or completeness for increased efficiency. Approximate reasoning algorithms can be tractable although the underlying language is not, in contrast to limiting attention only to inexpressive tractable fragments as e.g. [9].

* Research reported in this paper was supported by the German Federal Ministry Economics (BMWi) under the Theseus project, `http://theseus-programm.de` and EU project SEALS (FP7-ICT-238975), `http://www.seals-project.eu/`

Investigations into approximate reasoning usually start from a sound and complete algorithm and system and directly addresses performance bottlenecks in order to improve efficiency, i.e. the algorithms are altered, leading to approximate outputs, while improving speed and keeping the introduced error ratio as low as possible.

In previous work [10, 11] we have shown how to approximate instance retrieval for named classes within the KAON2 approach. In this paper we show how instance retrieval for complex classes can be approximated by reducing it to instance retrieval for named classes. For this, we compute what we call *approximate extensions* of complex classes by means of combining extensions of named classes, e.g. by using standard database operations. The approach leads to a speedup of about factor 10, while the number of introduced errors varies depending on the query, but is within reasonable bounds.

The present paper is structured as follows. After recalling necessary preliminaries, we will present our approach to approximate instance retrieval, and then describe our approximate algorithms. We conclude by reporting on corresponding evaluation results and with ideas for further work. For a more elaborate version of this paper we refer to our technical report [16].

2 Preliminaries

Description Logics. Description logics (DLs) are a family of knowledge representation formalisms. The basic constituents to represent knowledge in DLs are *concepts* C, *roles* r and individuals a. They are used to form *axioms* collected in a *knowledge base* KB to make statements about a domain of interest. We primarily consider the DL $\mathcal{SHIQ}$ with concept and role assertion axioms of the form $C(a)$, $r(a, b)$ that assign an individual to a concept or relate two individuals via a role, and concept inclusion axioms of the form $C \sqsubseteq D$ that state subclass relationships. For a detailed presentation of DLs we refer to [12]. The *signature* of a knowledge base KB, denoted by $\sigma(KB)$, is the set of all individual, concept and role names that occur in the axioms within KB. In particular, $\sigma(KB)$ comprises all individuals occurring in KB.

Instance retrieval with DL knowledge bases builds on the standard reasoning task of instance checking. An individual $a \in \sigma(KB)$ is an instance of a concept C with respect to a knowledge base KB if the axiom $C(a)$ is a logical consequence of KB, which is denoted by $KB \models C(a)$. Instance retrieval can be interpreted as the repeated application of instance checking for all known individuals of KB and a given concept. We call the result of retrieving all instances of concept C from KB the *(conventional) extension* of C with respect to KB, denoted by $|C|$, and define it as follows.[3]

$$|C| := \{x \in \sigma(KB) \mid KB \models C(x)\}$$

In the context of instance retrieval, the concept C is often also called the *query*.

[3] Notice that this notion of extension refers to a particular knowledge base and is different from the model-theoretic notion of extension defined for an interpretation.

Relational Algebra. Relational Algebra is the formal underpinning of modern relational database systems and is used to formalise database operations on the relational model originally introduced by Codd [13]. The main construct for representing data in the relational model is a *relation*, denoted by $R(a_1, \ldots, a_n)$, that represents a database table with column attributes a_1 to a_n and rows that instantiate the columns as tuples of values. Relational algebra expressions are used to formulate queries on the thus represented database tables and result themselves in relations, such that expressions can be nested. Attributes in a relation can be referred to by means of path expressions of the form $R\,.\,a_i$, e.g. within conditions.

We briefly recall the relational operators that are used in this paper. A *projection* $\pi_{[a_1,\ldots,a_m]}(R(a_1, \ldots, a_n))$ restricts the columns of the resulting relation to the attributes $a_1, \ldots, a_m$ for $m < n$. A *selection* $\sigma_{[\texttt{condition}]}(R(a_1, \ldots, a_n))$ selects those rows for which `condition` holds. A *cross product* $R_1(a_1, \ldots, a_n) \times R_2(b_1, \ldots, b_m)$ generates a combined relation $R(a_1, \ldots, a_n, b_1, \ldots b_m)$ in the sense of the Cartesian product by multiplying rows, which is used for join operations. Other set operations are used for relations as usual, namely *union* $R_1 \cup R_2$, *intersection* $R_1 \cap R_2$ and *difference* $R_1 \setminus R_2$, operating on relation tuples in the usual way. For a detailed description of relational algebra see e.g. [14].

3 Approximation of Instance Retrieval

Our approach for the approximation of instance retrieval queries is based on the notion of the *approximate extension* $\langle C \rangle$ of a concept C with respect to a knowledge base $\mathcal{KB}$. Intuitively, $\langle C \rangle$ is the set of instances that are obtained through interpreting complex concepts in C as simple set operations on the individuals known to $\mathcal{KB}$, starting from the atomic extensions of concepts and roles that occur in C. In this way, the model-theoretic semantics of DLs is approximated by a straightforward combination of results for atomic queries that requires less effort to compute than the reasoning process for complex instance retrieval queries in DLs does. The exact definition of an approximate extension is given in Table 1 recursively for all language constructs. For an example, consider the knowledge base $\mathcal{KB} = \{C \sqsubseteq A \sqcup B, A(a_1), C(a_2)\}$ and the instance retrieval query $A \sqcup B$. The conventional extension of the concept $A \sqcup B$ contains both individuals a_1 and a_2, i.e. $|A \sqcup B| = \{a_1, a_2\}$. However, the approximate extension of $A \sqcup B$ contains only a_1, i.e. $\langle A \sqcup B \rangle = \{a_1\}$.

The more complex the query concept C is, the more the approximate extension deviates from the conventional extension. For the simplest queries, such as atomic concepts, the two types of extensions coincide and no errors are made in instance retrieval. This characteristics is captured by the following proposition.

Proposition 1 (soundness and completeness of simple approximate extensions). *For a knowledge base $\mathcal{KB}$ and a concept C of the form $C = A_1 \sqcap \cdots \sqcap A_m \sqcap \neg B_1 \sqcap \ldots \neg B_n$, with all A_i and B_j atomic, the approximate extension of C is equivalent to its conventional extension, i.e. $\langle C \rangle = |C|$.*

Table 1. Definition of an approximate extension. A stands for atomic classes and C,D for complex (non-atomic) classes. R stands for roles and n for a natural number.

Approximate Extensions		
$\langle \top \rangle$	$=$	$\lvert \top \rvert$
$\langle \bot \rangle$	$=$	$\emptyset$
$\langle A \rangle$	$=$	$\lvert A \rvert$
$\langle \neg A \rangle$	$=$	$\lvert \neg A \rvert$
$\langle R \rangle$	$=$	$\{(x,y) \mid \mathcal{KB} \models r(x,y)\}$
$\langle R^- \rangle$	$=$	$\{(x,y) \mid \mathcal{KB} \models r(y,x)\}$
$\langle C \sqcap D \rangle$	$=$	$\langle C \rangle \cap \langle D \rangle$
$\langle C \sqcup D \rangle$	$=$	$\langle C \rangle \cup \langle D \rangle$
$\langle \neg C \rangle$	$=$	$\langle \top \rangle \setminus \langle C \rangle$
$\langle \exists R.C \rangle$	$=$	$\{x \in \langle \top \rangle \mid \exists y : (x,y) \in \langle r \rangle \wedge y \in \langle C \rangle\}$
$\langle \forall R.C \rangle$	$=$	$\{x \in \langle \top \rangle \mid \forall y : (x,y) \in \langle r \rangle \rightarrow y \in \langle C \rangle\}$
$\langle \leq n\, R.C \rangle$	$=$	$\{x \in \langle \top \rangle \mid \#\{y \mid (x,y) \in \langle r \rangle \wedge y \in \langle C \rangle\} \leq n\}$
$\langle \geq n\, R.C \rangle$	$=$	$\{x \in \langle \top \rangle \mid \#\{y \mid (x,y) \in \langle r \rangle \wedge y \in \langle C \rangle\} \geq n\}$

Proposition 1 states that, for queries that have the form of conjunctions of possibly negated named concepts, the approximate and conventional extensions have exactly the same instances. In other words, computing the approximate extension is *sound and complete* with respect to the conventional extension.

For more complex queries, however, the approximation might deviate significantly from the correct answer in both that it might miss instances as well as show improper instances. In particular the approximation of the complement constructor is supposed to cause significant deviation as it interprets negation in a closed-world sense, potentially including improper instances in an answer. Hence, we aim at eliminating general complements by means of normalisation, avoiding this source of error.

For standard reasoning in DLs a query concept can be expressed in various normal forms and semantics-preserving transformations do not affect the result of instance retrieval. For the calculation of approximate extensions, however, the result depends on the form of the concept, and different semantically equivalent concept expressions can have different approximate extensions. We can exploit this characteristics by choosing a normal form for query concepts that fits best the process of approximation in terms of both error rate and ease of computation. In this light, we consider the negation normal form [15] of concept expressions for queries, denoted by $\mathsf{NNF}(C)$ for a concept C, in which negation symbols are pushed inside to occur only in front of atomic concepts. This eliminates the case of considering the approximation for general complements with its rather drastic closed-world interpretation. Besides the lower expected error rate this also avoids the computationally costly handling of large sets of individuals in case of large ABoxes by an algorithm that computes approximate extensions. The positive effect that elimination of complement approximation has on the error rate in instance retrieval can be expressed by the following property, which ensures that

approximation of concepts in negation formal form only gives up completeness but preserves soundness at least for a certain class of queries.

Proposition 2 (soundness of limited approximate instance retrieval). *Let KB be a knowledge base and C be a concept such that $\mathsf{NNF}(C)$ contains no $\forall$- and no $\leq$- and $\geq$-constructs. The approximate extension of $\mathsf{NNF}(C)$ only contains instances that are also contained in the conventional extension of C with respect to KB, i.e. $\langle \mathsf{NNF}(C) \rangle \subseteq |C|$.*

Proposition 2 states that, for queries that do not make use of the $\forall$, $\geq$ and $\leq$ constructs (after normalisation), the approach of approximating concepts in their negation normal form yields an extension that might miss some instances but has no improper instances in it. In other words, computing the approximate extension is *sound* with respect to the conventional extension.

4 Computing Approximate Extensions

In this section, we will present algorithms for computing the approximate extension of a query concept. We will lay out the architecture of a system for approximate instance retrieval and elaborate on two implementations of the algorithms, one in a database and one in memory.

4.1 System Architecture

Our system for approximate instance retrieval takes as input a $\mathcal{SHIQ}$[4] knowledge base KB and a complex query concept Q to compute the approximate extension of Q with respect to KB as a set of individuals. This is depicted on the right-hand side of Figure 1. The principle behind computing $\langle Q \rangle$ is always to start from the individuals in the conventional extensions of (possibly negated) atomic concepts and (possibly inverse) roles that occur in Q and to recursively combine these according to the structure of concepts in Q, reflecting the set operations from Table 1. According to Propositions 1 and 2, this results in an answer that is sound and complete for some cases, only sound for others, or neither sound nor complete, depending on the language constructs used in the query.

We have implemented the approximate instance retrieval method in two different ways and distinguish between *database* and *in-memory* computation: in the first case computation is delegated to underlying database operations, whereas in the second case it is performed in main memory. While for the database variant the atomic extensions are pre-computed prior to query-time and materialised in the database using a sound and complete reasoner, the in-memory variant allows for two possibilities to access the atomic extensions: in *online processing* a sound and complete DL reasoner is invoked at query-time to compute

[4] We use $\mathcal{SHIQ}$ since we build on KAON2 for our experimental results. However, our approximation approach can easily be extended to nominals, the missing feature for handling OWL ontologies.

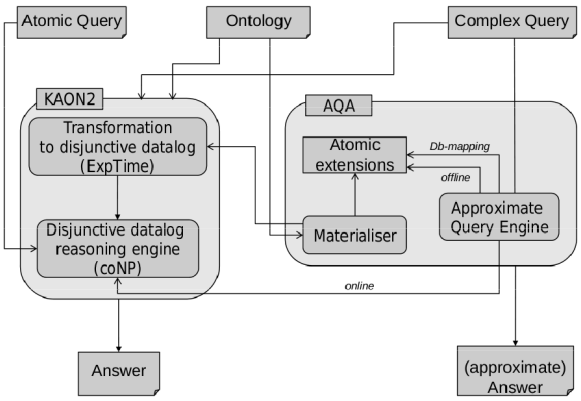

Fig. 1. An overview of the system architecture

atomic extensions, while in *offline processing* they are again pre-computed and materialised either in a database or in memory if possible. Database computation and offline processing are very useful when dealing with large amounts of data in scenarios with frequent querying on rather static ontologies, for which materialisation can be done in advance. Online processing is intended to be used in such cases where a materialisation is hardly manageable as ontologies are subject to frequent changes.

For online processing we utilise the KAON2 reasoner, as illustrated in Figure 1. The reason for this choice is that KAON2 was designed to be an efficient ABox reasoner on knowledge bases with large ABoxes and simple TBoxes in comparison to other state-of-art DL reasoners, which typically perform better on knowledge bases with large (or complex) TBoxes and small ABoxes. As depicted on the left-hand side of Figure 1, KAON2 transforms the TBox together with complex queries into a disjunctive datalog program in a first step, to perform ABox reasoning in a second step based on the result of this transformation. Hence, for every complex ABox query KAON2 needs to repeatedly perform the TBox transformation, which is computationally costly. For ABox queries that have the form of atomic concepts, however, this transformation is not necessary and can be bypassed. For the variant with in-memory and online processing we can take advantage of this because for computing atomic extensions with KAON2 the costly TBox translation is saved.

4.2 Delegation of Computation to Database

The variant that performs database computation is a presumably efficient implementation of approximate instance retrieval as the pre-computed atomic extensions are materialised and approximate extensions are computed by making use of highly optimised database operations. This variant is essential in practice for handling ontologies with large ABoxes that cannot be processed efficiently

Concept Expression	*Relational Algebra Expression*
$\tau_{\mathsf{db}}(A)$	$\pi_{[ind]}(\sigma_{[class=A]}(\mathsf{Ext}^C))$
$\tau_{\mathsf{db}}(\neg A)$	$\pi_{[ind]}(\sigma_{[class=\neg A]}(\mathsf{Ext}^C))$
$\tau_{\mathsf{db}}(\exists r.C)$	$E_C := \tau_{\mathsf{db}}(C)$ $E_r := \sigma_{[role=r]}(\mathsf{Ext}^r)$ $\pi_{[ind_1]}(\sigma_{[ind=ind_2]}(E_C \times E_r))$
$\tau_{\mathsf{db}}(\forall r.C)$	$E_C := \tau_{\mathsf{db}}(C)$ $E_r := \sigma_{[role=r]}(\mathsf{Ext}^r)$ $E_{\_} := \pi_{[ind_1]}(\sigma_{[ind \neq ind2]}(E_C \times E_r))$ $\pi_{[ind_1]}(\mathsf{Ext}^C) \setminus E_{\_}$
$\tau_{\mathsf{db}}(\leq n\, R.C)$	$E_C := \tau_{\mathsf{db}}(C)$ $E_r := \sigma_{[role=r]}(\mathsf{Ext}^r)$ $\pi_{[ind]}(\sigma_{[count(ind_1) \leq n \ \wedge \ ind=ind2]}(E_C \times E_r))$
$\tau_{\mathsf{db}}(\geq n\, R.C)$	$E_C := \tau_{\mathsf{db}}(C)$ $E_r := \sigma_{[role=r]}(\mathsf{Ext}^r)$ $\pi_{[ind]}(\sigma_{[count(ind_1) \geq n \ \wedge \ ind=ind2]}(E_C \times E_r))$
$\tau_{\mathsf{db}}(C_0 \sqcap C_1 \sqcap \cdots \sqcap C_n)$	$\tau_{\mathsf{db}}(C_0) \cap \tau_{\mathsf{db}}(C_1) \cap \cdots \cap \tau_{\mathsf{db}}(C_n)$
$\tau_{\mathsf{db}}(C_0 \sqcup C_1 \sqcup \cdots \sqcup C_n)$	$\tau_{\mathsf{db}}(C_0) \cup \tau_{\mathsf{db}}(C_1) \cup \cdots \cup \tau_{\mathsf{db}}(C_n)$

Table 2. Mapping of DL concept expression to Relational Algebra Expression

in memory. Here, the recursive combination of atomic extensions in terms of set operations as defined in Table 1 is completely delegated to the underlying database, which benefits performance. As a basis for this form of computation we use a database schema that consists of two Relations, namely $\mathsf{Ext}^C(ind, class)$ for storing concept extensions and $\mathsf{Ext}^r(ind_1, role, ind_2)$ for storing role extensions. In their schema, the attribute $ind_{(i)}$ stands for individual names, $class$ for names of possibly negated concepts and $role$ for names of possibly inverse roles. Starting from a knowledge base $\mathcal{KB}$, these two relations are initialised as follows.

$$\mathsf{Ext}^C(ind, class) = \{(a, C) \mid \mathcal{KB} \models C(a)\}, \ \textit{for } C = A \mid \neg A \ \textit{with } A \in \sigma(\mathcal{KB})$$
$$\mathsf{Ext}^r(ind_1, role, ind_2) = \{(a, r, b) \mid \mathcal{KB} \models r(a, b)\}, \ \textit{for } r = p \mid p^- \ \textit{with } p \in \sigma(\mathcal{KB})$$

Notice that, for the purpose of approximate instance retrieval, Ext^C and Ext^r form a complete representation of the original knowledge base $\mathcal{KB}$.

A complex query concept Q is answered by transforming its negation normal form $\mathsf{NNF}(Q)$ into a relational algebra expression according to a mapping τ_{db} and posed as a query to the underlying database system. The complete mapping definition for τ_{db} is given in Table 2. The left-hand side shows the concept constructors that can occur in $\mathsf{NNF}(Q)$ and the right-hand side shows their respective relational algebra expression. Recursive application of τ_{db} ultimately produces a single database query $\tau_{\mathsf{db}}(\mathsf{NNF}(Q))$ that is used for computing $\langle Q \rangle$.

For an example consider the query $Q = A \sqcap \exists r.\neg B$. The mapping τ_{db} produces the following nested relational algebra expression.

$$\tau_{\mathsf{db}}(Q) = \pi_{[ind]}(\sigma_{[class=A]}(\mathsf{Ext}^C)) \cap \\ \pi_{[ind_1]}(\sigma_{[ind=ind_2]}(\sigma_{[role=r]}(\mathsf{Ext}^r) \times \pi_{[ind]}(\sigma_{[class=\neg B]}(\mathsf{Ext}^C)))) \ .$$

When posed to the underlying database, this rather large expression is subject to efficient internal query optimisation strategies as they are typically employed by database systems.

4.3 In-memory Computation

Both the variants with online and offline processing share the same implementation of the approximate algorithm. The difference is the handling of atomic extensions which is presented by an additional function. This function takes as parameters a knowledge base and an atomic concept or atomic role for which the atomic extension is to be computed while the actual algorithm accepts the knowledge base and a complex concept query for which the approximate extension is to be computed. For the computation of the atomic extension, depending on the chosen variant, the introduced function invokes either a complete and sound reasoner or retrieves the atomic extension from the database. For the exact details of this algorithm, the interested reader may refer to [16].

5 Conclusion

In our experiments, using the WINE ontology, which has been designed as a showcase for the expressivity of OWL, we compared our algorithms with KAON2 as a sound and complete DL reasoner. Running the approximation algorithm in the database variant, we obtained a significant performance improvement for each $\exists$-query[5] about 90%. Running the algorithm in offline processing where the approximation is computed in memory, we obtained another significant performance gain, indeed about 99% compared to KAON2. For the details of our experiments including complex queries, the interested reader may refer to [16].

We have presented an approach to approximate instance retrieval based on approximate extensions. Compared with a complete and sound DL reasoner, our approach can significantly improve the performance of reasoning over expressive ontologies with large ABoxes and TBoxes. We presented several instantiations of our approach resulting online and offline in-memory and database variants. We evaluated the approaches and showed that a significant speed-up of around 90% can be obtained while the number of introduced errors remains relatively small.

Future work includes improvements on the online variant using logic programming engines, further experiments for complex queries, combinations with other approximate reasoning methods, extension to more expressive language features and applications of our approach in suitable use case scenarios.

[5] queries of the form $\exists r.A$ where A is a named class and r is a role

References

1. Schmidt-Schauß, M., Smolka, G.: Attributive Concept Descriptions with Complements. Artificial Intelligence **48**(1) (1991) 1–26
2. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web (2007) 51–53
3. Haarslev, V., Möller, R.: Racer System Description. In: IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning, London, UK, Springer-Verlag (2001) 701–706
4. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universität Karlsruhe (2006)
5. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM – A Pragmatic Semantic Repository for OWL. In Gil, Y., et al., eds.: Web Information Systems Engineering WISE 2005 Workshops, October 2005. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2005) 182–192
6. Tobies, S.: Complexity Results and Practical Algorithms for Logics in Knowledge Representation. PhD thesis, RWTH Aachen, Germany (2001)
7. Fensel, D., van Harmelen, F.: Unifying reasoning and search to web scale. IEEE Internet Computing **11**(2) (2007) 96, 94–95
8. Rudolph, S., Tserendorj, T., Hitzler, P.: What is Approximate Reasoning? In Calvanese, D., Lausen, G., eds.: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (RR2008). Lecture Notes in Computer Science, Springer (2008) 150–164
9. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ Envelope. In Kaelbling, L.P., Saffiotti, A., eds.: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, Professional Book Center (2005) 364–369
10. Hitzler, P., Vrandecic, D.: Resolution-Based Approximate Reasoning for OWL DL. In Gil, Y., et al., eds.: Proceedings of the 4th International Semantic Web Conference, Galway, Ireland, November 2005. Lecture Notes in Computer Science, Springer, Berlin (2005) 383–397
11. Tserendorj, T., Rudolph, S., Krötzsch, M., Hitzler, P.: Approximate OWL-Reasoning with Screech. In Calvanese, D., Lausen, G., eds.: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (RR2008). Lecture Notes in Computer Science, Springer (2008) 165–180
12. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2007)
13. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Commun. ACM **26**(1) (1983) 64–69
14. Ramakrishnan, R., Gehrke, J.: Database Management Systems. Osborne/McGraw-Hill, Berkeley, CA, USA (2000)
15. Schmidt-Schauß;, M., Smolka, G.: Attributive Concept Descriptions with Complements. Journal of Artificial Intelligence **48**(1) (1991) 1–26
16. Tserendorj, T., Grimm, S., Hitzler, P.: Approximate Instance Retrieval on Ontologies. Technical report, Kno.e.sis Center, Wright State University, Dayton, Ohio (2010) `http://knoesis.wright.edu/faculty/pascal/resources/publications/aqa2010.pdf`.