LAS: Extending Racer by a Large Abox Store

CuiMing Chen, Volker Haarslev, JiaoYue Wang
Concordia University, Montreal, Quebec, Canada
{cui_chen|haarslev|jiaoy_wa}@cse.concordia.ca

Abstract

Recently, several approaches have been proposed on combining description logic (DL) reasoning with database techniques. In this paper we report on the LAS (Large Abox Store) system extending the DL reasoner Racer with a database used to store and query Tbox and Abox information. LAS stores for given knowledge bases their taxonomy and their complete Abox in its database. The Aboxes may contain role assertions. LAS can answer Tbox and Abox queries by combining SQL queries with DL reasoning. The architecture of LAS is based on merging techniques for so-called individual pseudo models.

1 Introduction

Description Logics (DLs) historically evolved from a combination of frame based systems and predicate logic. The semantic organization of data and the powerful deductive capabilities of DL reasoners are its main characteristics. Due to the recent use of DL reasoners for OWL-DL and the availability of large data sets in the semantic web, research was stimulated on merging DL reasoners with databases, which are well-known for their efficient data management of huge data sets [8].

Thus, these distinct features between description logics and databases have led to research on how to make good use of their respective advantages. As early as in 1983, [12] already distinguished two ways for a reasoning system to obtain data from a database. In 1993, an approach on loading data into a DL reasoner was investigated [1]. A later proposal [2] extends a traditional DL Abox by a DBox so that users can transparently ask queries without being concerned about what DB or KB has to be accessed. In [8] it was pointed out that the previous approaches were lacking of an automated translation between DL and database schemas. It proposed an object oriented model and two translations so that a schema in this model can be translated into a description logics schema as well as a database schema.

Another significant application that employs databases to support reasoning over a large number of individuals is the Instance Store [3]. It uses a database to store a Tbox taxonomy and Abox individual assertions and reduces Abox reasoning to terminological reasoning by querying a traditional relational database. Although effective, it has some limitations because it can only deal with role-free Aboxes [7].

In our paper, we present an approach that employs relational database techniques as a filter for DL reasoners. It utilizes the data stored in a database and SQL queries for (partially) answering Abox queries. In case of incomplete information, simplified queries are forwarded to Racer.

2 Background of LAS: Large Abox Store

In this section, we will briefly describe the scope of our system and the techniques employed in LAS.

2.1 The $\mathcal{ALCH}_{\mathcal{R}^+}$ Language

We assume that the reader is familiar with $\mathcal{ALC}$ [4]. The logic $\mathcal{ALCH}_{\mathcal{R}^+}$, which extends $\mathcal{ALC}$ by adding role hierarchies and transitively closed roles, is the DL supported by LAS. On the one hand, the added expressiveness leads to more complicated algorithms for dealing with Abox queries. On the other hand, since we do not allow number restrictions and functional roles, relationships between individuals are either asserted or can be easily inferred (in case a role is transitive or has super-roles).

2.2 Techniques to deal with role assertions in Aboxes

2.2.1 Precompletion techniques

A well-known technique to reason with role assertions in Aboxes is called precompletion [6]. By applying so-called precompletion rules, role assertions can be eliminated and individuals become independent of one another. Thus, Abox reasoning can be reduced to standard Tbox reasoning.

As proven in [9] for the DL $\mathcal{SHF}$, the precompletion algorithm will always terminate no matter which of the applicable rules is chosen first. Although different strategies for the priority of rules to be chosen can lead to a different computing complexity, as long as disjunctions of concepts exist, the worst case of the computational complexity will be exponential. Moreover, the characteristics of the precompletion technique may cause an exponential number of precompletions and could inhibit essential optimization techniques such as dependency-directed backtracking.

2.2.2 Pseudo model techniques

Another solution to reasoning with Aboxes containing role assertions is called individual pseudo model merging [5, 10]. Individual pseudo models are derived from the initial Abox satisfiability test and exploited for various Abox reasoning tasks. For instance, the test whether an individual is not an instance of a concept might be replaced by a so-called pseudo model mergability test. This test is motivated by the observation that an individual is usually an instance of only a few named concepts. The pseudo model technique reuses information cached in pseudo models. It can be considered as a sound but incomplete structural test possibly avoiding "expensive" Abox satisfiability tests. For instance, if the pseudo models of an individual i in an Abox $\mathcal{A}$ and a concept $\neg C$ do not interact, one can safely conclude that i is not an instance of C. If the test returns false, one has to test the satisfiability of $\mathcal{A} \cup \{i : \neg C\}$ (see [5, 10] for more details).

2.2.3 Our choice: pseudo model techniques

The advantage of the precompletion algorithm is that it can reduce Abox reasoning to Tbox reasoning. However, due to the existence of disjunctions, possibly an exponential number of precompletions can be generated. As a result, in case of a clash, one has to prove other precompletions. Compared to the precompletion technique, although the pseudo model merging technique is sound but incomplete, its minimal computational overhead and the avoidance of any indeterminism outweigh its incompleteness, especially when efficiency is the main concern for a system. On the other hand, besides using the sound but incomplete merging technique, LAS always forwards queries it can't resolve to RACER, so the final answer would still be sound and complete.

3 Design and Architecture of LAS

The LAS system implements several standard Abox inference techniques through a collection of SQL queries exploiting DBMS optimization facilities for dealing with large amounts of data. Concerning Tbox queries, LAS merely retrieves the taxonomy from RACER and employs standard SQL for querying the Tbox hierarchy. As for Abox reasoning, LAS retrieves from Racer pseudo models for all individuals and named concepts and stores this information in the database. The pseudo model merging test is implemented via SQL. It is used to reduce the number of candidates for Abox queries by filtering out individuals that are proven to be not relevant for a particular query. In case of incomplete information for fully answering a query LAS still relies on RACER's query facilities to answer this query. However, it forwards to RACER only the reduced set of individuals relevant to answer this query. In the presence of thousands of individuals, the time savings can be significant. LAS store every information retrieved from RACER in order to efficiently answer future queries. This approach is especially efficient for the old data (from a previous run of RACER) because queries might be answered easily without restarting RACER.

3.1 Architecture

The Large Abox Store is a Java application. It consists of four components:

- an ontology, such as OWL-DL or RACER file,
- a reasoner–RACER,
- a database–such as Oracle 9i, which can be accessed through JDBC,
- a user interface.

3.2 Database Schema

We defined a database schema for storing the taxonomy, (individual) pseudo models, concept membership for and role relationships between individuals in database tables. Due to space limitations, we briefly describe only a small part (used to store the Abox) of this schema.

Ind (indn, indpmid, indcl)
IndPModel (indpmid, ina, innota, inest, inusl, unique)

InAssertion (indn, desn, mstspf)
RoleAssertion (ind1n, ind2n, rn, cpl1, cpl2, rcpl)

In table Ind, we store the individual name, the pseudo model ID of the individual, and a status flag 'indcl' to indicate the completeness of the individual information. If it is complete, it means that we know all the concept names of which this individual is an instance. The individual's pseudo model, defined as the tuple $\langle M^{+}, M^{\neg A}, M^{\exists}, M^{\forall} \rangle$ [5], is stored in the table IndPModel. The status flag 'unique' is to describe the uniqueness of the pseudo model for this individual. A pseudo model is unique if it does not depend on a disjunction. The table InAssertion stores individual assertions, where 'desn' refers to a concept description ID and 'mstspf' to the most specific concept names of which the individual is an instance. In the table RoleAssertion, the 'cpl1' status flag indicates that all fillers of 'ind1n' for role 'rn' are known (it can be used in query-individual-fillers); 'cpl2' indicates that all direct predecessors of a 'ind2n' for role 'rn' are known (it can be used in query-direct-predecessors); 'rcpl' asserts that all for related individual pairs for the role 'rn' are known. For a role assertion (ind1, ind2, rn), we call ind2 a filler of ind1 for role rn and ind1 a predecessor of ind2; ind1 and ind2 are related individuals w.r.t. role rn.

3.3 Implemented queries

LAS covers most of the basic Tbox and Abox queries provided by RACER.

3.3.1 Tbox queries

Tbox queries can be divided into two parts: queries about named and complex concepts. Since we store all information about named concepts and roles as well as their parents and children in the database, the information regarding the taxonomy and role hierarchy is complete. As a result, when a user poses queries about named concepts, the system will just execute a SQL query instead of querying RACER.

Concerning subsumption queries about complex concepts, LAS will retrieve the pseudo models of complex concepts from RACER and subsequently use them for mergability tests. Using the mergability test for queries (e.g., parents of a complex concept), LAS usually can eliminate non-subsumers. However, the remaining candidates are still not the final result. LAS has to call RACER to verify the remaining set of possible subsumers. One reason is that RACER might provide only one non-unique pseudo model even though the concept is based on disjunctions. The other reason is that LAS only deals with the flat but not deep pseudo models, which means the mergability test is too conservative for existential and universal role restrictions.

3.3.2 Abox queries

The implemented Abox queries comprise the following services:

- most-specific named concepts for a given individual,
- all individuals that are instances of given concept,
- role fillers, direct predecessors,
- roles relating individuals,

- individuals pairs related via a role.

Due to lack space, we only discuss the implementation of query_retrieve as a sample. For this query, we distinguish between named and complex concepts because LAS stores only the pseudo models of named concept in its tables. However, if a query contains a complex concept the corresponding pseudo model is retrieved from RACER and used to execute the query. Afterwards, this pseudo model is discarded in order to avoid an potential blow-up of the tables. The pseudo code from query_retrieve is as follows.

```
Query_retrieve (String concept, String concept_status, Reasoning reasoner,
                Connection c)
   {
    if (concept_status == "atomic"){
        descpl= "SELECT Des.descpl FROM Description WHERE Des.desn ='concept'"
        //information of the concept is completed
        if (descpl== True)
        retrieved_indivdiuals="SELECT DISTINCT indn FROM InAssertion
                               WHERE desn ='concept'"
        else {
            //do the mergable test and get the possible candidates
            Vector individual_candidates = mergabletest (String concept);
            //send back the candidates for RACER to do the final check
            retrieved_indivdiuals = reasoner.concept_instances(concept, abox,
                                                    individual_candidates)
            //store back the result and set the mstspt status flag TRUE
            store (retrieved_individuals, Connection c, Table InAssertion)
            update(Table Des, String descpl, String ind);
            }
    }else if (concept_status == "complex"){
        //negate the complex concept and get this negation's pseudo model
        String neg_concept = "(not " +concept+ ")";
        Vector des_neg_psmodel = reasoner.get_cnp_psmodel(neg_concept);
        Vector individual_candidates = mergabletest (String concept);
        retrieved_indivdiuals = reasoner.concept_instances(concept, abox,
                                                 individual_candidates)
        }
    return retrieved_individuals;
    }
```

Now we give the mergability test based on SQL.

```
 mergable_test (String concept)
    {
     define candidates as ResultSet;
     define sql1 as
     "select distinct Ind.indn from Ind
     where Ind.indpmid in
     (select IndPmodel.indpmid from IndPModel,Des,DesPModel
     where Des.desn = 'concept' and Des.negdespmid=DesPModel.despmid and
     ((IndPModel.ina = DesPModel.desnot and IndPModel.ina <> 'NIL') or
```

```
  (IndPModel.innota = DesPModel.des and IndPModel.innota <> 'NIL') or
  (IndPModel.inest = DesPModel.desusl and IndPModel.inest <> 'NIL') or
  (IndPModel.inusl = DesPModel.desext and IndPModel.inusl <> 'NIL')))";"
 candidates = c.execute(sql1)"
 return candidates;
}
```

Role assertion queries: Role assertion queries are implemented in two ways according to the system mode. For the lazy mode, LAS relies completely on RACER. For the eager mode, it uses SQL to generate the complete information for the posed query. For example, for the query to retrieve all role fillers of an individual, it first finds all the descendants of the given role, and then generates the transitive closure, if the role is transitive. At last, it extract the results from the newly generated or updated RoleAssertion table.

3.4 Evaluation of test results

In order to test the performance of the LAS System, we used the OWL benchmark from [11] (university ontology) as an experiment. The university ontology contains a sufficient number of role assertions and can be easily generated in increasing sizes.

Figure 1 lists the runtimes for loading the benchmark data and storing all the basic information into the LAS system. The times are given in seconds. The columns show from left to right the number of universities (all with only 1 department per university), the number of individuals, concept and role assertions, and finally the LAS load time. The Tbox remains unchanged but the number of universities is increased from 1 to 10 resulting in increasing Abox sizes. The runtimes show a roughly linear increase w.r.t. the Abox size.

No. Un	No. inds	concept assert	role assert	LAS LTime
1	1554	1656	4114	72
2	2920	3171	7935	180
3	4079	4498	11128	276
4	5184	5773	14157	388
5	6325	7126	17243	502

No. Un	No. inds	concept assert	role assert	LAS LTime
6	7447	8487	20573	702
7	8620	9881	11128	932
8	9585	11083	26738	1058
9	12950	15279	36793	1613
10	14089	16709	40222	1681

Figure 1: Scalability of LAS loading time (in seconds).

Based on this ontology, we designed a set of Abox queries. We use for this benchmark 1 university (1,656 individual concept assertions), 5 universities (11,083 individual concept assertions,) and 10 universities (16,709 individual concept assertions) for testing. The first three queries (see the table on the left in Figure 2) are (i) compute all named concepts of which an individual is an instance (Ind types), (ii) compute only the most specific named concepts of which an individual is an instance (Ind drt types), (iii) retrieve all individuals that are instances of a given concept (rtr ind). All queries performed are divided into two groups: the initial use (new) where LAS has to load

the ontology file and initialize the database and the repeated use (old) where the ontology has been unchanged and a query can answered using the already filled tables of the database. For all queries LAS is used in two different modes: lazy and eager (see Section 3.3.2 above). The columns in the first table show from left to right the query type, the number of universities, usage mode, size of answer set, time used by RACER (without using LAS), time used by LAS (including RACER's time).

The second set of three queries (see the table on the right in Figure 2) is about (iv) compute the role fillers of individuals (Ind fillers), (v) compute the direct role predecessors of individuals (Ind direct prdec), (vi) compute all pairs of individuals related via a given role (rel ind). The second table has two columns for the runtimes of LAS, one for the eager and one for the lazy mode.

The time taken to query the individual types and individual direct types in LAS are slightly longer than for RACER alone because in both queries LAS has to ask RACER initially for the result. However, for retrieving individuals, due to the pseudo model techniques, LAS is faster than RACER alone. Moreover, if RACER is terminated, it will take the same time to load the ontology and process the query again, but in LAS the answer can be retrieved instantly.

For the role assertion queries one can see that employing the eager mode of LAS mostly reduces the query time because the communication with and reasoning of RACER is significantly reduced.

Query	#U	use mod	anw size	RACER alone	LAS
Ind types	U1	new	5	17.6	18.1
		old		0.001	0.012
	U5	new	5	488	644
		old		0.001	0.1
	U10	new	5	898	923
		old		0.01	0.67
Ind drt types	U1	new	1	18.5	22.1
		old		0.001	0.08
	U5	new	1	492	503
		old		0.001	0.03
	U10	new	1	960	1104
		old		0.01	0.23
rtr ind	U1	new	28	19.3	2.30
		old		0.98	0.16
	U5	new	214	526	12.4
		old		0.05	0.39
	U10	new	298	834	32.8
		old		0.15	0.48

Query	#U	use mod	anw size	RACER alone	LAS lazy	LAS eager
Ind fillers	U1	new	1	18.2	19	1.76
		old		0.001	0.06	0.07
	U5	new	1	400	496	11.7
		old		0.02	0.04	0.44
	U10	new	1	946	1314	21.2
		old		0.01	0.50	0.55
Ind direct prdec	U1	new	1	15.4	17.3	1.6
		old		0.01	0.06	0.07
	U5	new	1	495	654	1.17
		old		0.02	0.53	0.4
	U10	new	1	699	887	14.6
		old		0.01	0.5	0.55
rel ind	U1	new	1878	18.7	24.9	1.72
		old		0.002	0.19	0.17
	U5	new	2089	466	513	5.39
		old		0.035	0.17	0.25
	U10	new	2929	687	1031	11.4
		old		0.02	0.2	0.12

Figure 2: Concept/Role assertion query time (in seconds) and answer size.

4 Conclusion and Future Work

In this paper we presented the LAS system, a description logic repository that extends a relational database with description logic inference capabilities. The main feature of our system is that it can deal with Abox role assertions. Acting as a filter for RACER, it speeds up several Abox queries. For some of the queries, although LAS might be a bit slower for the first time than RACER alone, it can extract the results instantly for repeated uses no matter whether RACER has been terminated or not. Moreover, LAS is not constrained by the Unique Name Assumption (UNA), it can deal with situations where individuals can have different names.

The future work of our system includes to explore more optimizations so that it can support more complex queries. So far we only used the Oracle database for developing and testing our system. We also plan to update our system such that it can be used with more relational databases.

References

[1] A.Borgida and R.Brachman. Loading data into description reasoners. volume 22. SIGMOD, 1993.

[2] Paolo Bresciani. Querying database from description logics. In *KRDB'95*, 1995.

[3] D.Turi. *Instance Store*. http://instancestore.man.ac.uk/, 2004.

[4] F.Baader, D.Calvanese, D.Nardi, P.F, and Patel-Schneider. *The Description Logic Handbook: Theory, Implememntation, and Applications*. Cambridge University Press, 2003.

[5] Volker Haarslev and Ralf Möller. Optimizing TBox and ABox reasoning with pseudo models. In *Proceedings of the International Workshop in Description Logics 2000 (DL2000)*, 2000.

[6] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. In *Ann. of Mathematics and Artificial Intelligence*, 1996.

[7] I.Horrocks, L.Li, D.Turi, and S.Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Description Logic Workshop*, 2004.

[8] M.Roger, A.Simonet, and M.Simonet. Bringing together description logics and database in an object oriented model. In *DEXA2002*, 2002.

[9] Sergio Tessaris and Ian Horrocks. Abox satisfiability reduced to terminological reasoning in expressive description logics. In *Logic for Programming, LPAR 2002*, 2002.

[10] V.Haarslev, R.Möller, and A.Turhan. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In *Proceedings of International Joint Conference on Automated Reasoning*, 2001.

[11] Y.Guo, J.Heflin, and Zhengxiang Pan. Benchmarking DAML+OIL repositories. In *Second International Semantic Web Conference, ISWC 2003, LNCS 2870*, 2003.

[12] Y.Vassiliou, J.Clifford, and M.Jarke. How does an expert system get its data? In *VLDB*, 1983.