Proceedings of the 4th Workshop on

New Forms of Reasoning for the Semantic Web: Scalable & Dynamic

Stefano Ceri, Emanuele Della Valle, Jim Hendler, and Zhisheng Huang (Eds.)

NeFoRS 2010

May 31th, 2010, Heraklion, Greece

http://nefors10.larkc.eu/

Program

Date: May 31st, 2010

9:00 - 9:05 Introduction

9:05 - 9:45 *Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus* (Invited talk)
Stream Reasoning: Where We Got So Far

9:45 - 10:20 *Thorsten Liebig, Andreas Steigmiller and Olaf Noppens,*
Scalability via Parallelization of OWL Reasoning

10:20-10:45 Break

10:45-11: 20 *Aidan Hogan, Axel Polleres, Jürgen Umbrich and Antoine Zimmermann*
Some entities are more equal than others: statistical methods to consolidate Linked Data

11:20-11:55 *Raffael Stein and Valentin Zacharias*
RDF on Cloud Number Nine

11:55 - 12:30 *Nikolaos Papadakis, Polydoros Petrakis and Haridimos Kondylakis*
A Solution to the Ramification Problem Expressed in Temporal Description Logics

12:30 Close

Preface

Initiatives like Linked Open Data have resulted in a rapid growth of the Web of data, and this growth is expected to continue. While impressive progress has been made in recent years in scalable storing, querying, and reasoning with languages like RDFS and OWL, existing reasoning techniques fail to perform when applied at Web-scale, due to the quantities of instance data, expressiveness of the ontologies, or the inherent inconsistency and incompleteness of data on the Web.

These problems of scale are increasingly compounded by the appearance of highly dynamic data streams. Data streams occur in modern applications such as traffic engineering, applications of RFID tags, telecom call recording, medical record management, financial applications, and clickstreams. On the Web, many sites distribute and present information in real-time streams of semi-structured text. In many of these application areas, the ability to perform complex reasoning tasks that combine streaming information (both data and text) with background knowledge would be of great benefit. Stream reasoning is a new multidisciplinary approach for semantically processing high-frequency high-volume streams of information in combination with rich background knowledge.

The NeFoRS2010 workshop is a joint continuation of earlier successful workshop on scalable and dynamic reasoning for the Semantic Web, NeFoRS'07, NeFoRS'08, and SR'09.

The workshop organizers would like to thank the following researchers for their involvement and contribution to the work of the programme committee: Daniele Braga (Politecnico di Milano, Italy), Irene Celino (CEFRIEL, Italy), Marko Grobelnik (Josef Stefan Institute, Slovenia), Michael Grossniklaus (Politecnico di Milano, Italy and ETH Zurich, Switzerland), Pascal Hitzler (Wright State University, Ohio, USA), Mihai Lupu (Information Retrieval Facility, Austria), Marko Luther(DOCOMO Research Labs, Munich, Germany), Vassil Momtchev (Ontotext, Bulgaria), Jose Quesada (Max Planck Institute, Germany), Angus Roberts (University of Sheffield, United Kingdom), Magnus Sahlgren (SICS, Sweden), Anne Schlicht (University of Mannheim, Germany), Stefan Schlobach (Free University of Amsterdam, the Netherlands), Lael Schooler (Max Planck Institute, Germany), Volker Tresp, (SIEMENS, Germany), Giovanni Tummarello (DERI, Ireland), and Michael Witbrock (CYC Europe).

The NeFoRS2010 PC Chairs

Stefano Ceri (Politecnico di Milano, Italy)
Emanuele Della Valle (Politecnico di Milano, Italy),
Jim Hendler (Rensselaer Polytechnic Institute, USA)
Zhisheng Huang (Free University of Amsterdam, the Netherlands)

Stream Reasoning: Where We Got So Far

Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus

Dip. di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
email: {dbarbieri, braga, ceri, dellavalle, grossniklaus } @elet.polimi.it

Abstract. Data Streams - unbounded sequences of time-varying data elements - are pervasive. They occur in a variety of modern applications including the Web where blogs, feeds, and microblogs are increasingly adopted to distribute and present information in real-time streams. We foresee the need for languages, tools and methodologies for representing, managing and reasoning on data streams for the Semantic Web. We collectively name those research chapters Stream Reasoning. In this extended abstract, we motivate the need for investigating Steam Reasoning; we characterize the notion of Stream Reasoning; we report the results obtained by Politecnico di Milano in studying Stream Reasoning from 2008 to 2010; and we close the paper with a short review of the related works and some outlooks.

1 Motivation

The use of the Internet as a major source of information has created new challenges for computer science and has let to significant innovation in areas such as databases, information retrieval and semantic technologies. Currently, we are facing another major change in the way information in provided. Traditionally information used to be mostly static with changes being the exception rather than the rule. Nowadays, more and more dynamic information, which used to be hidden inside dedicated systems, is getting available to decision makers. A large part of this dynamic information is an (almost) "continuous" flow of information with the recent information being more relevant as it describes the current state of a dynamic system.

Continuous processing of these flows of information (namely data streams) has been largely investigated in the database community [1]. Specialized Data Stream Management Systems (DSMS) are available on the market and features of DSMS are appearing also in major database products, such as Oracle and DB2.

On the contrary, continuous processing of data streams *together with rich background knowledge* requires specialized reasoners, but work on semantic technologies is still focusing on rather static data. In existing work on logical reasoning, the knowledge base is always assumed to be static (or slowly evolving). There is work on changing beliefs on the basis of new observations [2], but the

solutions proposed in this area are far too complex to be applicable to gigantic data streams of the kind illustrated in the oil production example above.

As argued in [3], we strongly believe that there is a need to close this gap between existing solutions for belief update and the actual needs of supporting decision process based on data streams and rich background knowledge. We named this little explored, yet high-impact research area Stream Reasoning.

2 Stream Reasoning

In this section we characterize the notion of Stream Reasoning giving a definition and explaining what is peculiar to it.

Definition 1. ***Stream Reasoning****: logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.*

Peculiar to stream processing, and thus also to Stream Reasoning, are the notions of window [4] and continuous processing [5]. In the following we characterize Stream Reasoning with regards to these two notions.

Window Traditional reasoning problems are based on the idea that all the information available should be taken in to account when solving the problem. In Stream Reasoning, we eliminate this principle and restrict reasoning to a certain window of concern which consists of a subset of statement recently observed on the stream while previous information is ignored. This is necessary for different reasons. First of all, ignoring older statements allows us to saves computing resources in terms of memory and processing time to react to important events in real time. Further, in many real-time applications there is a silent assumption that older information becomes irrelevant at some point.

Continuous Processing Traditional reasoning approaches are based on the idea that the reasoning process has a well defined beginning (when a request is posed to the reasoner) and end (when the result is delivered by the system). In Stream Reasoning, we move from this traditional model to a continuous processing model, where requests in terms of reasoning goals are registered at the reasoner and are continuously evaluated against a knowledge base that is constantly changing.

3 Current State of Development in Politecnico di Milano

Deductive Stream Reasoning has been studied at Politecnico di Milano. In [6], we specified a general and flexible architecture - based on the LarKC conceptual architecture [ICSC08] - for reasoning in the presence of data streams and rich background knowledge. This architecture leverages existing DSMS and SPARQL engines and anticipates the possibility to extend RDF graphs, by introducing

RDF streams and extending SPARQL to continuously processing RDF streams observed through windows. Both these extensions are present in Continuous SPARQL (or simply C-SPARQL) [7].

C-SPARQL adds RDF streams to the data types supported by SPARQL, much in the same way in which the stream type has been introduced to extend relations in relational data stream management systems. An RDF stream is defined as an ordered sequence of pairs, where each pair is made of an RDF triple and its timestamp τ:

$$\ldots$$
$$(\langle subj_i, pred_i, obj_i \rangle\ ,\ \tau_i)$$
$$(\langle subj_{i+1}, pred_{i+1}, obj_{i+1} \rangle\ ,\ \tau_{i+1})$$
$$\ldots$$

Timestamps can be considered as *annotations* of RDF triples, and are monotonically non-decreasing in the stream ($\tau_i \leq \tau_{i+1}$). More precisely, timestamps are not strictly increasing because they are not required to be unique. Any (unbounded, though finite) number of consecutive triples can have the same timestamp, meaning that they "occur" at the same time, although sequenced in the stream according to some positional order.

Consider the program querying Twitter for tweets containing "I'm reading". The result is a stream that get continuously update with new tweets that match the search criteria. Such stream can be represented as an RDF stream of triples, stating what each twitter user is reading, annotate with the posting date-time.

$$\ldots$$
$$(\langle :Giulia, :isReading, :Twilight \rangle, \text{2010-02-12T13:34:41})$$
$$(\langle :John, :isReading, :TheLordOfTheRings \rangle, \text{2010-02-12T13:36:28})$$
$$\ldots$$

As a simple example of C-SPARQL, we informally explain a query that counts how many followers of Giulia have been reading a book in the last hour.

```
1 REGISTER QUERY NumberOfGiuliaFollowersWhoAreReadingBooks COMPUTE EVERY 15m
      AS
2 SELECT count(distinct ?user) as ?numberOfGiuliaFollowersReadingBooks
3 FROM <http://streamingsocialdata.org/followersNetwork>
4 FROM STREAM <http://streamingsocialdata.org/reading>
5                     [RANGE 1h STEP 15m]
6 WHERE { ?user :follows :Giulia.
7         ?user :isReading ?x .
8         ?x a :Book . }
```

At line 1, the REGISTER clause is use to tell the C-SPARQL engine that it should register a continuous query, i.e. a query that will continuously compute answers to the query. The COMPUTE EVERY clause states the frequency of every new computation, in the example every 15 minutes. The query joins background and streaming knowledge. At line 3, the standard SPARQL clause FROM is used to load in the default graph an RDF graphs describing who is following who on Twitter. At line 4, the clause FROM STREAM is used to tell the C-SPARQL engine to process the stream exemplified above. Next, line 5 defines the window of observation over the RDF stream. The window considers all the stream triples in the last hour, and is advanced every 15 minutes. The content of the window is loaded in the default graph as if it were a standard RDF graph.

However, every 15 minutes new triples enter into the window and old triples exit from the window, thus the default graph is modified accordingly. Note that the query result does not change during the slide interval, and is only updated at every slide change. Triples arriving in the stream between these points in time are queued until the next slide change and do not contribute to the result until then. The WHERE clause is standard; it includes a set of matching patterns that operates over the default graph as in a standard SPARQL query. The results are project by the SELECT clause at line 2, which also count the number of distinct bindings of the variable ?user.

In [8], we propose a formal semantics of C-SPARQL language together with a query graph model which is an intermediate representation of queries devoted to optimization. We discuss the features of an execution environment, based on [6] that leverages existing technologies and we introduce optimizations in terms of rewriting rules applied to the query graph model, so as to efficiently exploit the execution environment.

Finally in [9], we elaborate on the deductive reasoning support to C-SPARQL. The [7] version of C-SPARQL can work under entailment regimes different from RDF simple semantics, but at the cost of recomputing, when the window slides, any deduction that depends from the triples in the window. In [9], we propose a technique for incremental maintenance of materializations of ontological entailments that exploits the transient nature of streaming data. By adding expiration time information to each RDF triple, we show that it is possible to compute a new complete and correct materialization whenever the window slides, by dropping explicit statements and entailments that are expired, and then incrementally adding all the deduction that depends on the new triples that entered the window and tagging them with a correct expiration time.

The following example extends the C-SPARQL query above by requesting to count only the followers of Giulia who were only twittering about books. The ontological definition of user, who only twittered about books, is given in OWL-RL as follows:

```
:UserOnlyTwitteringAboutBooks rdfs:subClassOf :User;
  rdfs:subClassOf [
          a owl:Restriction;
          owl:onProperty :tweets;
          owl:allValuesFrom :Book;
  ] .
```

An example of C-SPARQL query that leverages such ontological definition can be the following one:

```
1 REGISTER QUERY NumberOfGiuliaFollowersWhoAreReadingBooks COMPUTE EVERY 15m
      AS
2 SELECT count(distinct ?user) as ?numberOfGiuliaFollowersOnlyReadingBooks
3 FROM <http://streamingsocialdata.org/followersNetwork>
4 FROM STREAM <http://streamingsocialdata.org/reading>
5                     [RANGE 1h STEP 15m]
6 WHERE { ?user :follows :Giulia.
7         ?user a :UserOnlyTwitteringAboutBooks .
8         ?user :isReading ?x .
9         ?x a :Book . }
```

In order to be evaluated, this C-SPARQL query requires the engine to reason about the triple in the window and incrementally evaluate which users satisfy the ontological definition of UserOnlyTwitteringAboutBooks.

4 Related Work

Two approaches, alternative C-SPARQL exists: Streaming SPARQL [10] and Time-Annotated SPARQL (or simply TA-SPARQL) [11]. Both languages introduce the concept of window over stream, but only C-SPARQL brings the notion of continuous processing, typical of stream processing, into the language; all the other proposal still rely on permanent storing the stream before processing it using one-shot queries. Moreover, only C-SPARQL proposes an extension to SPARQL to support aggregate functions[1]. Such a support for aggregates is designed specifically to exploit optimization techniques [8] that push, whenever possible, aggregates computation as close as possible to the raw data streams.

We believe that C-SPARQL is an important piece of the Stream Reasoning puzzle, it supports OWL RL entailment [9] is only a first step toward Stream Reasoning.

There are many ideas of how to support incremental reasoning on the different levels of complexity. In particular, there are approaches for the incremental maintenance of materialized views in logic [13], object-oriented [14] and graph databases [15], for extensions of the well known RETE algorithm for incremental rule-based reasoning [16, 17] and even some initial ideas of how to support incremental reasoning in description logics [18, 19]. All of these methods operate incrementally, thus they are appropriate to treat information that changes, but none of them is explicitly dedicated to process an (almost) continuous flow of information with the recent information being more relevant.

5 Outlook

In these two years of work on Stream Reasoning, we have been mainly working on C-SPARQL and its corresponding infrastructure. We believe they provide an excellent starting point for Stream Reasoning research, because a) RDF streams provide an RDF-based representation of heterogeneous streams and b) C-SPARQL construct query can create RDF snapshots that can feed information into existing reasoning mechanisms. We already made a first step in this direction, investigating the incremental maintenance of ontological entailment materializations [9]. This approach needs to be generalized to more expressive ontological languages.

Moreover, the extraction of patterns from data streams is subject of ongoing research in machine learning. For instance, results from statistical relational learning are able to derive classification rules from example data in very effective

[1] For a comparison between C-SPARQL and SPARQL 1.1 support for aggregates see [12]

ways. In our future work, we intend to link relational learning methods with C-SPARQL to facilitate pattern extraction on top of RDF streams.

Finally, we envision the possibility to leverage recent developments in distributed and parallel reasoning [20, 21] as well as the compilation of reasoning tasks into queries [22] for scaling up to gigantic data streams and to extremely large numbers of concurrent reasoning tasks.

Acknowledgements

The work described in this paper is has been partially supported by the European project LarKC (FP7-215535).

References

1. Garofalakis, M., Gehrke, J., Rastogi, R.: Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
2. Gaerdenfors, P., ed.: Belief Revision. Cambridge University Press (2003)
3. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems **24**(6) (2009) 83–89
4. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: STREAM: The Stanford Stream Data Manager (Demonstration Description). In: Proc. ACM Intl. Conf. on Management of data (SIGMOD 2003). (2003) 665
5. Babu, S., Widom, J.: Continuous Queries over Data Streams. SIGMOD Rec. **30**(3) (2001) 109–120
6. Della Valle, E., Ceri, S., Barbieri, D.F., Braga, D., Campi, A.: A First Step Towards Stream Reasoning. In: Proc. Future Internet Symposium (FIS). (2008) 72–81
7. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-sparql: Sparql for continuous querying. In: WWW. (2009) 1061–1062
8. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An Execution Environment for C-SPARQL Queries. In: Proc. Intl. Conf. on Extending Database Technology (EDBT). (2010)
9. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In Aroyo, L., Antoniou, G., Hyvonen, E., eds.: ESWC. Lecture Notes in Computer Science, Springer (2010)
10. Bolles, A., Grawunder, M., Jacobi, J.: Streaming SPARQL – Extending SPARQL to Process Data Streams. In: Proc. Europ. Semantic Web Conf. (ESWC). (2008) 448–462
11. Rodriguez, A., McGrath, R., Liu, Y., Myers, J.: Semantic Management of Streaming Data. In: Proc. Intl. Workshop on Semantic Sensor Networks (SSN). (2009)
12. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Feedbacks on sparql 1.1 support for aggregates. Technical report, LarKC Project. (February 2010) Available on line at http://wiki.larkc.eu/c-sparql/sparql11-feedback.
13. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. J. Data Semantics **2** (2005) 1–34

14. Kuno, H., Rundensteiner, E.: Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. IEEE Transactions on Data and Knowledge Engineering **10**(5) (september 1998) 768–792
15. Zhuge, Y., Garcia-Molina, H.: Graph structured views and their incremental maintenance. In: Proceedings of the Fourteenth International Conference on Data Engineering. (1998) 116 – 125
16. Fabret, F., Regnier, M., Simon, E.: An adaptive algorithm for incremental evaluation of production rules in databases. In: Proceedings of the 19th International Conference on Very Large Data Bases. (1993) 455 – 466
17. Berster, B.: Extending the rete algorithm for event management. In: Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02). (2002)
18. Cuenca-Grau, B., Halaschek-Wiener, C., Kazakov, Y.: History matters: Incremental ontology reasoning using modules. In: Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007. Volume 4825 of Lecture Notes in Computer Science. (2007) 183–196
19. Parsia, B., Halaschek-Wiener, C., Sirin, E.: Towards incremental reasoning through updates in owl-dl. In: Proceedings of the Reasoning on the Web Workshop at WWW2006, Edinburgh, UK. (2006)
20. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.: Owl reasoning with webpie: calculating the closure of 100 billion triples. In: Proceedings of ESWC 2010, Heraklion, Crete. (2010)
21. Schlicht, A., Stuckenschmidt, H.: Distributed resolution for expressive ontology networks. In: Web Reasoning and Rule Systems. (2009)
22. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic el using a relational database system. In: IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2009) 2070–2075

RDF on Cloud Number Nine

Raffael Stein[1] and Valentin Zacharias[2]

[1] Karlsruhe Institute of Technology (KIT), Karlsruhe Service Research Institute (KSRI), Englerstr. 11, 76131 Karlsruhe, Germany
raffael.stein@kit.edu
[2] Forschungszentrum Informatik (FZI), Haid-und-Neustr. 10-14, 76131, Germany
zacharias@fzi.de

Abstract. We examine whether the existing 'Database in the Cloud' service SimpleDB can be used as a back end to quickly and reliably store RDF data for massive parallel access. Towards this end we have implemented 'Stratustore', an RDF store which acts as a back end for the Jena Semantic Web framework and stores its data within the SimpleDB. We used the Berlin SPARQL Benchmark to evaluate our solution and compare it to state of the art triple stores. Our results show that for certain simple queries and many parallel accesses such a solution can have a higher throughput than state of the art triple stores. However, due to the very limited expressiveness of SimpleDB's query language, more complex queries run multiple orders of magnitude slower than the state of the art and would require special indexes. Our results point to the need for more complex database services as well as the need for robust, possible query dependent index techniques for RDF.

1 Introduction

The current trend of Cloud Computing promises to make complex IT solutions available as service to customers in a way that is elastically scalable and priced with a utility pricing model that enables customers to only pay for what they need. For the customer cloud services are expected to bring the following advantages: 1) The customer does not need to worry about the complexities of running the respective IT solution 2) Large cloud computing companies might be able to realize scale effects and hence offer a solution at a price lower than would be possible at a smaller scale 3) The customer does not need to provision infrastructure for an estimated future peak demand because he/she can scale the solution on demand and pay only for what is needed.

Many different kinds of cloud computing services are currently offered, ranging from infrastructure services (that allow to rent virtual computers) to complex software service [1]. Of particular interest for this paper are "database as a service" offerings - and here in particular the probably oldest and best established one: SimpleDB by Amazon [2]. This service offers access to a distributed, scalable and redundant database management system with a utility pricing model. For the user they hold the promise of not having to worry about the problematic

issues of a large scale database development; of not having to worry about redundancy, backups, and scaling to large data sets and many concurrent database clients. This paper examines whether these promises can be realized for the storage of RDF; whether the SimpleDB service allows to build a scalable, "worry free" triple store.

We have examined this question by creating Stratustore, a system that extends the Jena Semantic Web Framework to store the data in the SimpleDB. We have evaluated its performance using the Berlin SPARQL Benchmark.

After a short discussion of related work this paper starts with a description of the most important properties of the SimpleDB service. Next, the architecture and design of the Stratustore is described. Then the evaluation setup and results are detailed before a conclusion and discussion.

This paper is a summary presentation of the diploma thesis by Raffael Stein which has been published in German under the title "Entwicklung eines skalierbaren Semantic Web Datanspeichers in der Cloud".

2 Related Work

To the authors best knowledge no comparable work that uses and evaluates a database cloud service as a back end for a triple store exist. There is, however, considerable work showing great promise for the use of column stores - the database technology also underlying the SimpleDB service - for the storage and querying of RDF data [3–5]. Under the name of "Connected Commons" [6], Talis is offering specifically a triple store as a service. This service is surely an important offer if only RDF data needs to be processed, however, it lacks the maturity and entire ecosystem of different services to be found with Amazon.

Amazon Web Services

Amazon was one of the first large companies to provide cloud computing services and by now offers a large variation of cloud services. For the realization and evaluation of the Stratustore we used the Simple Storage Service (S3), the Elastic Compute Cloud (EC2) and - most importantly - the SimpleDB. Each of these service will be introduced below.

The **Simple Storage Service (S3)** is a distributed key-value store that scales to very large data sets and massive parallel access. The use of the service is charged based on traffic and the amount of data stored. Within the context of this work it was used mostly to store the machine images that were then executed with EC2 (see below). The S3 is also needed for large CLOBS that go beyond the limits of SimpleDB (see below).

The **Elastic Compute Cloud (EC2)** is a service that offers virtual computer instances that can be rented on an hourly basis. These instances are started with an operating system and applications based on an machine image stored in the S3. EC2 is charged based on the properties of the virtual computer (the customer can choose between different numbers of cores, sizes of RAM and hard

disks), the time it is used and the data transferred into and out of Amazon's data centers. In the context of this work EC2 was used to run all tests and evaluations - indeed the assumption behind Stratustore is, that it would be used mostly by applications that are already executed within Amazons cloud.

The **SimpleDB** is a distributed database that is accessed through a simple REST or SOAP interface. The use of the SimpleDB is charged based on the amount of data stored (on a much higher rate than for the S3) and based on the computing time needed for query processing.

The SimpleDB service works without a user defined schema and organizes data in the following way (see also figure 1):

- At the highest level data is organized in *domains*, which can be thought of as akin to a collection or a table.
- Within domains data is stored in *items*, which is the basic unit of data for the SimpleDB.
- Each item then has a number of *attributes* each of which can have one or more *values*.

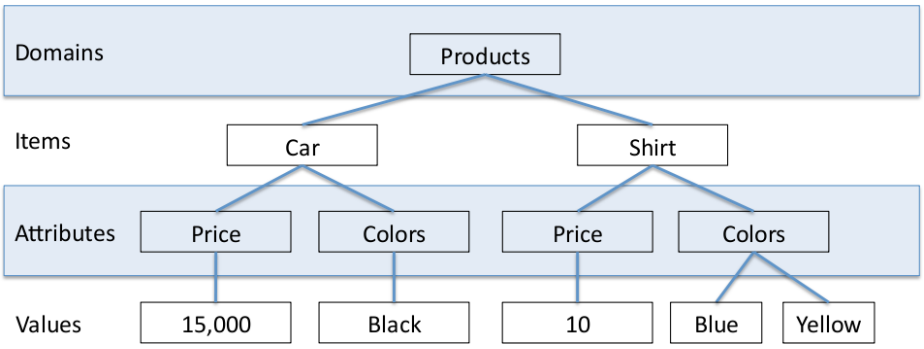


Fig. 1. Data organization inside the SimpleDB

A number of limitations exist that restrict the flexibility of this data organization: A domain must not be larger than 10GB, an item must not have more than 256 attribute-value pairs and an attribute value may not be larger than 1024 bytes. The SimpleDB also uses strings as the only data type - meaning that all comparisons have to be done lexically and that numbers have to be padded with zeros to make this possible.

The SimpleDB keeps multiple copies of a user's data to ensure reliability and security of the data. However, the way this is done means that it supports only "eventual consistency" [7] where the propagation of changes can take a few milliseconds and a user query may hit an out-of-date copy. It is not even guaranteed that a query after a write by the same client will reflect that write operation. SimpleDB also lacks any support for normalized data.

The biggest drawback is the limited expressiveness of SimpleDB's query language. In syntax this language is modeled after SQL select queries, it is, however, much less expressive. In particular it does not allow for comparisons between stored entities; it does not allow for any kind of join operation. That means queries containing dependencies must be split into several parts. Imagine asking the database for all work colleagues of Bob. We do not know the company name, so a first query has to find out the company Bob works at. A second query can then find everyone who works at that company. Without an index for that specific case, it is not possible to answer a query like this in a single run.

3 Stratustore

We have implemented the Stratustore that extends the Jena Semantic Web Framework such that it can use Amazon's SimpleDB as back end (in addition to relational database and main memory back ends already provided). A sketch of its overall architecture is shown in figure 2.

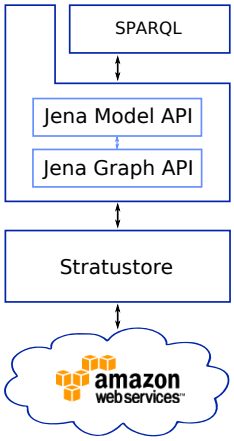


Fig. 2. Architectural overview of the Stratustore

The Stratustore is realized as an implementation of the Jena Graph API which supports a simple interface organized around triples. A detailed description of the layers of Jena can be found in [8]. Adding and removing is done on a per triple basis and queries are realized based on TripleMatch objects; objects that specify the expected values for some or all of the parameters of a triple. The data models of RDF and of the SimpleDB differ considerably and there are many different ways to map the one to the other.

The easiest mapping is a triple oriented mapping - one triple is mapped to one item, with the attributes subject, predicate and object. This simple mapping

avoids SimpleDB's restrictions with respect to the maximum attribute/value count. However, this mapping also means that - because of the missing join functionality in SimpleDB's query language - only a very small part of SPARQL query processing can be done by the SimpleDB. Most of the processing for queries needs to be done on the client resulting in a large amount of data that needs to be transferred over the network and very bad performance.

Other mappings define sets of triples that are jointly represented by one item. For example all triples that share the same subject and predicate can be represented in one item. Or all triples that share the same predicate could be mapped to one item. These mappings increase the portion of queries that can be answered directly within SimpleDB, however, at the same time they increase the risk of hitting the limits of the SimpleDB data model.

For the Stratustore we settled for an entity oriented mapping: here one item represents the data known about one subject, i.e. one item for entity "s" contains the data from all triples that contain s as a subject. Each item has an attribute "S" that has the URI of the subject represented by the current item. The other attributes then each represent one predicate defined for this subject and the attribute values represent the objects (recall that multiple values for one attribute are allowed in SimpleDB's data model). Taking figure 1 as an example, that means that a shirt is stored as a item with the attributes "S" containing the subject URI, the attribute "price" containing the price and the attribute "colors" containing the available colors.

The entity oriented mapping allows to push a larger proportion of SPARQL queries into the SimpleDB, for many cases entire SPARQL queries can now be translated into SimpleDB queries. However, it runs afoul of SimpleDB's limitations on the number of attribute-value pairs: Whenever an entity appears as subject in more than 255 triples there will need to be more than one item and queries will have to be split. Simple techniques can restrict the runtime impact occasional split items have; nevertheless it adds considerable additional complexity. Another problem for the entity oriented mapping is the fact that SimpleDB does not allows to query for attribute names, and that hence with the entity oriented mapping variables in predicate position pose a challenge. This second problem can be solved simply by creating a second item for each entity, this time with attribute names representing object and attribute values representing predicates, however, this leads to a doubling of storage requirements and update times. The final problem (affecting all possible mappings) is caused by the size limitation on attribute values which have no correspondence in RDF. This means that long values have to be stored separately in the S3 and that a separate text index is necessary. The code to handle these restrictions of the SimpleDB is not currently implemented in Stratustore and so it too is bound by these restrictions. The Berlin SPARQL Benchmark does not break these restrictions, so this was (almost) no problem for the evaluation.

3.1 Uploading Data

By the time this work was performed (early 2009), SimpleDB did not support any bulk uploading and hence naive uploading (one triple at a time) took a very long time. We used 1) triples collected by subject, 2) sorted inputs and 3) multi-threading to help tackle this problem.

1. Since multiple data changes to one item can be bundled into one request and since one item represents all the triples known about one subject URI, the Stratustore merges consecutive triple update requests with the same URI. This means that it holds back and combines consecutive update requests for triples that have the same subject. The item thus constructed is sent to the SimpleDB only when an update request with a different URI arrives or the the update process is finished.
2. To make best use of this bundling based on consecutive triples we pre-sorted the input triples by subject for larger updates.
3. Finally we used 10 threads running in parallel. More than 10 threads and distributing the uploading over more than one machine brought no additional benefit - 10 threads on one computer were already already able to saturate the write capacity of one SimpleDB domain.

Recently the SimpleDB interface was changed to allow bulk uploads of up to 25 items at one time. With this new functionality we would expect the upload speed to increase by up to one order of magnitude, the structure of our implementation would not need to change fundamentally.

3.2 Querying Data

The Stratustore supports querying through the Jena Graph API and the SPARQL language. SPARQL is the W3C standardized query language for RDF graphs. Its specification can be found at [9]. The triple oriented query operations of the Graph API can be translated directly into SimpleDB SELECT queries. To answer SPARQL queries, a mapping from SPARQL to the much simpler SELECT language is needed. The SPARQL query strings are broken into parts which are then transformed into SELECT queries and evaluated separately by SimpleDB. The results of all single queries are merged again and returned to the user. For this the pipeline shown in figure 3 is used.

The SPARQL query is first received and parsed by the Jena Semantic Web framework. Sets of triple patterns are created from the query and handed (via the Graph API) to the Stratustore. The Stratustore then groups the triple patterns by subject. The grouping is done to take advantage of the entity oriented schema, which is used to store the triples inside SimpleDB. Each pattern group can be combined if the single patterns ask about the same subject. One SELECT query is created for each group of triple patterns and all of these are posed in parallel to the SimpleDB. An example of the conversion of a SPARQL to a SELECT query is given in figures 4 and 5. The patterns which have the same subject form a SELECT query. The results of both SELECT queries have to be matched

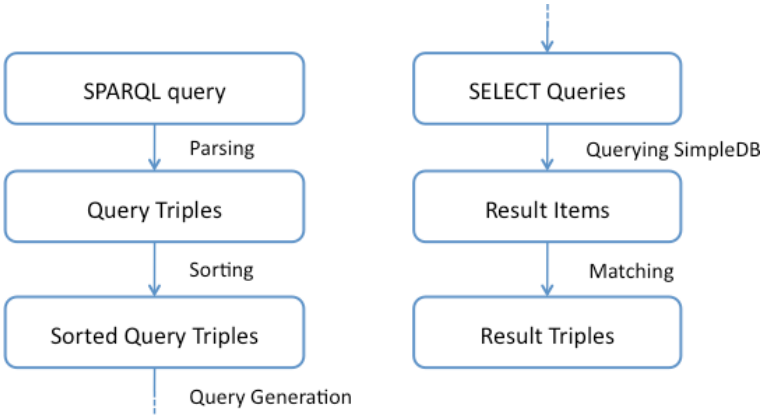


Fig. 3. Processing of a SPARQL query inside the Stratustore

against each other after they have been retrieved. It is not possible to compare two entries inside the SimpleDB. Therefore, two seperate queries are necessary. Query marshaling, cryptographic signing and remote access are handled by the Typica Library [10].

In response to these queries the SimpleDB returns the list of items matching the SELECT queries. In many cases multiple requests must be posed to retrieve all results for a SELECT query, since the size of responses is limited both in the number of items returned as well as the overall size of the message. Stratustore then re-constructs the triples based on the items and performs any necessary joins to find the variable assignments for the triple patterns and returns these to the broader Jena Framework.

Finally the Jena Framework applies any needed additional processing - this includes applying FILTER statements as well as combining graph group patterns, particular to realize the UNION statements. After this processing the result to the SPARQL query is returned to the user.

```
@prefix example: <http://example.org/> .
@prefix country: <http://downlode.org/rdf/iso-3166/countries#>

SELECT ?s WHERE {
  ?s rdf:type example:shirt .
  ?s example:producedBy ?m .
  ?m rdf:type example:Producer .
  ?m example:producesIn country:DE .
}
```

Fig. 4. Pattern Grouping inside the Stratustore

```
SELECT s, example:producedBy FROM products WHERE
  rdf:type = "example:shirt"

SELECT s FROM products WHERE
  rdf:type = "example:Producer"
  INTERSECTION
  example:producesIn = "country:DE"
```

Fig. 5. The aforementioned SPARQL query transformed into SimpleDB SELECT queries

4 Evaluation

We chose to use the Berlin SPARQL Benchmark (BSBM) [11] for evaluation. This gives us the possibility to compare the figures to other Semantic Web stores already tested with BSBM.

The BSBM is a benchmark system which allows the evaluation of semantic data stores that provide a SPARQL endpoint. It is based on an e-commerce use case which is centered around a set of products that are offered by different vendors. For the products, there are reviews written by consumers. To query the generated triples, the BSBM provides a sequence of 12 different SPARQL queries which simulate a user interaction with this e-commerce scenario.

The benchmark consists of a data generator and a query driver. The generator can be used to create sets of connected triples of any size. The query driver constructs queries which fit to the generated triples and executes them on the SPARQL endpoint. The queries and their constellation in the original benchmark can be found on the BSBM website at [12]. Response times are measured and logged.

The complete evaluation system was set up on Amazon's Elastic Compute Cloud (EC2) and is sketched in figure 6. One test system is always running on one EC2 instance (for the evaluation we used the default "Small Instance", see [13]). On each test system we ran both a BSBM test driver and the Stratustore.

To make the Stratustore accessible for the BSBM benchmark diver, Joseki[3] is used. Joseki is an HTTP server which offers a SPARQL endpoint and accesses RDF data via its interface to the Jena model. Behind the Jena model stands the Stratustore which in turn accesses the (remote) SimpleDB. This setup allowed to easily test the scalability in terms of parallel accesses of multiple users by starting up multiple instances of the test system. During evaluation it quickly became apparent, that two BSBM queries (2 and 5) have extremely long and useless runtimes and for time reasons we excluded them from most evaluation runs. A third query (query 11) had to be excluded, because it is not yet supported by

[3] `http://www.joseki.org/`

our implementation. We shortly detail the problem with each of these queries below.

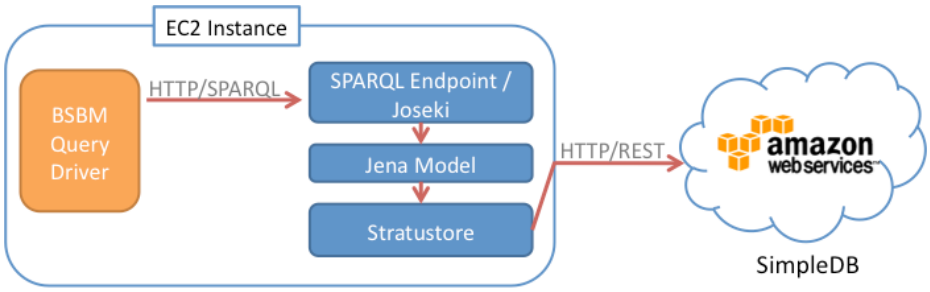


Fig. 6. The testing environment

Query 2 retrieves the labels of various subjects. The combination of query patterns of the same subject leads to a reordering of the patterns of this SPARQL query. This reordering however leads to a certain pattern not being as specific as before. In particular, not only some but all the labels of the whole database have to be retrieved. The combination of patterns is needed to take advantage of the entity oriented data model which the Stratustore is based on. In the particular case of query 2, the execution time is worsened by this behavior. A weighted sorting mechanism which prefers more specific query patterns and puts them first is thought to bring better results here.

Query 5 models a search for products with similar properties in the e-commerce use case. Naturally, this query touches a lot of data. It uses rather complex FILTER statements which, in the current implementation, cannot be mapped into SELECT queries. This forces the Stratustore to locally evaluate them which shows a bad performance. The solution to decrease the runtime of query 5 would be to push the interpretation of the FILTERs into the Stratustore.

The third query which had to be excluded from benchmarking is query 11. It contains a query statement with the the predicate as a variable. A query of the predicate is currently not supported in the Stratustore. A second index with predicates as the index key would be needed to cover this kind of query.

5 Results

Both EC2 and SimpleDB are hosted multi-tenancy services under constant development - hence it comes as no surprise, that runtimes fluctuated considerable in an unpredictable way. The fluctuations observed where within one order of magnitude. This has to be taken into account for the results reported below: albeit the BSBM benchmark ran over a considerable amount of time, actual

queries might run half as fast or with double the speed in a way unpredictable to us.

Garfinkel also experienced the unpredictability of Amazon's web services [15]. He states that Amazon has a very broad service level agreement with no throughput guarantees. Thus variations in performance can and do occur on all services.

Uploading a set of 1 million triples with a total size of 86MB, took as long as 144 minutes using only one thread. The use of multi threading speeded this up by a factor of 4.6. For 10 threads, the upload took 31 minutes to complete. This means, the transfer rate to the SimpleDB is about 47 kB (or 500 triples) per second. Adding more threads or more computers does not increase the transfer rate. This restriction was identified as a limitation of SimpleDB's acceptance rate. As mentioned before, bulk uploading is thought to bring an improvement.

A performance analysis of the Stratustore was done by running the query driver from the BSBM against the Stratustore. The data generator was used to create a data set of 1 million triples. These triples were then uploaded onto the Stratustore and queried against. However, we were not able to execute all of BSBM's queries against the Stratustore. The aforementioned three queries which had abnormal runtimes of several minutes were excluded from the benchmarking.

In the following we present the results of 9 different queries being run 50 times against a triple set of 1 million triples on the SimpleDB. The amount of executed queries per second (QpS) is evaluated. This metric is used in BSBM to compare the speed of execution of single queries.

In table 1, the runtimes of the different queries are presented. The row indicates which query was evaluated. The column indicates how many instances of the Stratustore were accessing the SimpleDB at the same time. Up to 20 simultaneous instances were tested. The figures in table 1 show the amount of queries which were successfully executed per second. The amounts of the single instances were added up for the total value in each field of the table.

Concurrent instances	1	5	10	20
Query 1	10.84	71.66	157.36	334.75
Query 3	1.99	10.77	22.89	48.27
Query 4	10.66	59.38	127.78	266.44
Query 6	0.28	1.31	3.00	6.24
Query 7	1.32	9.89	30.27	65.47
Query 8	0.81	6.42	22.56	47.01
Query 9	11.45	62.88	160.08	333.3
Query 10	1.94	11.33	27.07	57.2
Query 12	0.02	0.05	0.10	0.20

Table 1. Performance of queries of the Berlin SPARQL Benchmark in queries per second

When comparing the query per second rate of single queries to existing RDF stores like Jena[4], Virtuoso[5], Sesame[6] or Mulgara[7], the Stratustore is outperformed by multiple orders of magnitude. The authors of the BSBM evaluated these stores on [14]. The figures in the text used as comparison against the Stratustore are taken from these tests.

Examining for example query 6 with a QpS rate of 0.28 when executed on a single instance. The Virtuoso Store using a local database reaches a QpS rate of 55.0 in the same setting. The Virtuoso Store is about 2 magnitudes faster than the Stratustore. Comparing the fastest query on the Stratustore shows a similar outcome. On the Stratustore, query 1 reaches a speed of 10.84 queries per second, whereas the Virtuoso store processes 202.3 of this query per second. One of the main reasons for the lower performance of single query execution on the Stratustore can be found in the overhead of transforming SPARQL queries into queries of SimpleDB's SELECT language. The lower expressiveness of SimpleDB's query language requires a time intense restructuring of queries.

The strength of the Stratustore lies in the ability to easily scale in terms of users simultaneously accessing it. This can be seen in table 1 as well. Taking query 1 as an example again, when doubling the number of concurrent users from 5 to 10, the rate of processed queries per second increases by a factor of 2.2. Again doubling the concurrent accesses to 20, the QpS rate increases by a factor of 2.1. The Stratustore is able to answer the additional queries with no performance decrease. All queries are still answered at the same speed, despite the fact that the number of concurrent queries has doubled. The unexpected speed up of more than a factor of 2 is due to different loads of the SimpleDB on different times. Running a whole query set could take up to several hours thus we could not estimate the impact of different usage loads on the SimpleDB.

The BSBM contains results for the Virtuoso store running 8 and 64 clients simultaneously. For 8 clients, Virtuoso comes to 286.84 queries per second, while for 64 clients, the rate reaches 232.94 QpS. As can be seen from table 1, the Stratustore with 20 concurrent clients is able to reach a higher throughput than the Virtuoso store with as many as 64 clients. This shows that the Stratustore is able to provide the throughput of a state-of-the-art RDF store with the potential of scaling very well in terms of concurrent users.

6 Discussion and Future Work

Is the Stratustore on SimpleDB the "worry free" triple Store that frees the user from having to worry about the complexities of handling a distributed database; from having to worry about redundancy, backups and scaling to large data sets and many concurrent database clients?

[4] `http://jena.sourceforge.net/`
[5] `http://www.openlinksw.com/virtuoso`
[6] `http://www.openrdf.org`
[7] `http://www.mulgara.org/`

On the plus side it really is an instantly available distributed and replicated database that can serve many concurrent clients and that is payed for with a utility pricing model. For queries that make the best use of the SimpleDB data model it is - for a large number of concurrent accesses - even competitive to state of the art triples stores running on dedicated servers. For these simple queries throughput and response time is sufficient for interactive applications.

On the negative side, however, the Stratustore cannot store and query arbitrary RDF data, is very slow on more complex queries and it is inherently limited for transactional data. The following paragraphs detail these points.

As stated above the Stratustore is currently bound by some of SimpleDBs limitations. For this reason it currently cannot process queries with variables in predicate position, cannot store more than 255 triples involving the same subject and cannot store values larger than 1024 bytes. There are, however, simple workarounds for each of these limitations that could be implemented in Stratustore.

A more serious problem is the slow runtime for more complex queries; a slow runtime caused by joins having to be executed on the client side. Some optimizations are possible in Stratustore that could speed this up, for example some filter statements could be "pushed down" into the SimpleDB select queries - for some cases this could radically reduce the amount of data that needs to be transferred over the network and that needs to be joined. Another optimization would be the use of summary graphs or dataset statistics to arrive at better query plans that avoid transferring too large amounts of data. Finally, if the query types that will need to be answered are known in advance, then query dependent indexes (also stored in SimpleDB) could be used to tackle this problem.

Another issue is that the lack of complex database transactions together with the model of eventual consistency mean that the Stratustore cannot be used for use cases where transactions are important. In conclusion, the Stratustore is a solution for use cases that need a simple, robust, cheap, always on, mirrored database that is accessed concurrently with a read heavy work-load where simple queries are enough. For other applications the "worry free triple store" will need either indexes adapted to the applications need or a next generation, more powerful SimpleDB.

Lastly there is the question whether a "normal" database benchmark is appropriate for a cloud service or whether different benchmarks need to be designed. For example Binnig et al. outline how benchmarking Cloud applications differs from well established benchmarking systems [16]. Cloud services cannot be tested in a managed environment where all configuration parameters are under control. Instead, they are always exposed to unpredictable variations. This loss of control has to be taken into account when designing a benchmark. They argue that Cloud services ideally scale in a linear way with the amount of queries because more resources are added as needed to fulfill the requests. Their suggestion is to increase the amount of issued queries over time and count the successful interactions over a given amount of time. We partly followed this scheme by running different test cycles and increasing the number of clients.

References

1. Lenk, A., Klems, M., Nimis, J., Tai, S., Sandholm, T.: What's inside the Cloud? An architectural map of the Cloud landscape. In: CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, Washington, DC, USA, IEEE Computer Society (2009) 23–31
2. Amazon Web Services: SimpleDB. [Online] `http://aws.amazon.com/simpledb/`.
3. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable semantic web data management using vertical partitioning. In: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment (2007) 411–422
4. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In Wang, J.T.L., ed.: SIGMOD Conference, ACM (2008) 967–980
5. Sidirourgos, L., Goncalves, R., Kersten, M.L., Nes, N., Manegold, S.: Column-Store Support for RDF Data Management: not all swans are white. PVLDB **1**(2) (2008) 1553–1563
6. Talis: Talis connected commons. [Online] `http://www.talis.com/platform/cc/`.
7. Vogels, W.: Eventually consistent. Communications of the ACM **52**(1) (January 2009) 40–44
8. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters, New York, NY, USA, ACM (2004) 74–83
9. W3C: SPARQL Query Language for RDF. [Online] (2008) `http://www.w3.org/TR/rdf-sparql-query`.
10. Kavanagh, D.: Typica : A Java client library for a variety of Amazon Web Services. [Online] `http://code.google.com/p/typica/`.
11. Bizer, C., Schultz, A.: Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints. In: Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008). (2008)
12. Bizer, C., Schultz, A.: SPARQL Queries for the Berlin SPARQL benchmark. [Online] `http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/index.html#queriesTriple`.
13. Amazon Elastic Compute Cloud Developer Guide: Instance types. [Online] `http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/instance-types.html`.
14. Bizer, C., Schultz, A.: Berlin SPARQL Benchmark Results. [Online] `http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html`.
15. Garfinkel, S.: An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical report (2007)
16. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the Weather tomorrow?: Towards a Benchmark for the Cloud. In: DBTest. (2009)

A Solution to the Ramification Problem Expressed in Temporal Description Logics

Nikos Papadakis[1], Polydoros Petrakis[1], Dimitris Plexousakis[2],Haridimos Kondylakis[2]

1: Science Department, Technological Educational Institute of Crete
2: Computer Science Department University of Crete & Institute of Computer Science, FORTH,Crete
{ npapadak,ppetrak,dp,kondylak)@csd.uoc.gr

Abstract. The ramification problem in Artificial Intelligence is concerned with the indirect effects of an action. It has been shown in previous work that the ramification problem can be solved with the use of integrity constraints and actions representation. In this paper we begin with a quick review of the existing Description Logics Languages, and then we describe a Temporal Extension of Description Logics, able to represent integrity constraints, temporalized actions and non persistent effects. We describe a thorough solution to the ramification problem in Temporal Settings expressed in Temporal Description Logics.

Keywords: Ramification problem, Temporal Description Logics, Temporalized Actions

1 Introduction

The ramification problem, in Artificial Intelligence field, is concerned with the indirect effects of an action. In other words, it deals with the problem of representing the consequences of an action. It is a hard and ever existing problem, in systems exhibiting a dynamic behavior [11].

We describe a solution to the ramification problem, by using an example originally presented by [11]. However, the example and solution are this time expressed in whole, in Temporal Description Logics instead of First Order Logic. In order to accomplish that, we put together features of existing Description Logics languages and enrich them to become more specific and deterministic, so as to describe the integrity constraints and temporalized actions, as well as the effects of those actions, with uttermost clarity. More specifically, we combine features from the language $\mathcal{TL}$-$\mathcal{F}$ presented by Artale and Franconi [3], along with features of the syntax rules from the Schmiedel proposal [2] (also shown in figure 4), in order to be able to represent actions and integrity constraints, in Interval Based Temporal Description Logics. We are also able to represent the non persistent effects of those actions. The effects refer to specific and well defined time intervals.

In the following section (Section 2), we are going to present the basic syntax of Description Logics, along with existing implementations of Description Logic Languages and some Temporal Extensions of Descriptions Logics. In Section 3, we present a solution to the ramification problem with the use of an example and Temporal Description Logics. We provide algorithms for the production of static rules and the evaluation of dynamic and static rules. In the last part of Section 3, there are two theorems proving the correctness of the previously presented algorithms. In Section 4, we summarize the

information provided in this paper, and describe further extensions or applications of the provided solution of the ramification problem.

2 Description Logics Basics and Previous Work

In this section, we will initially present cases where it is useful to migrate from First Order Logic to Description Logics. Then we will describe the basics of Description Logics and mention existing Description Logic Languages, as well as Temporal Extensions to Description Logics.

First Order Logics is nowadays the most common way used for knowledge representation. However using FOL (First Order Logic) to represent Knowledge Bases in Relational Databases has proved to be inefficient, as FOL has too much expressive power and therefore lacks computational speed and efficient procedures. Also, Semantic Networks and frames do not require the whole part of First Order Logic. Additionally, the direct use of FOL can have too low inference power for expressing interesting theories. Therefore "Description Logics" has been introduced as a formal language for representing knowledge and reasoning about it [7] Description Logics, a structured fragment of FOL, is nowadays considered the most important knowledge representation formalism, unifying and giving a logical basis to Frame-based systems, Semantic Networks, Object Oriented and semantic data models. Description Logics are preferred for their high expressivity and decidability, as well as their reasoning algorithms, which always return correct answers [3]. It is shown by [8], that Knowledge Base satisfiability in Description Logics (specifically with language $\mathcal{ALCQI}$) can be EXPTIME-decidable and EXPTIME-complete.

The basic types of a concept language are concepts, roles, and individuals (concept names) [5]. A concept is a description of a collection of individuals with common properties [13], for example "Employee" is a concept. Roles express relations between these individuals of concepts. For example "supervise", represents the relationship between individuals belonging in Employee and Employer concepts. Individuals are constants of concepts e.g. "Nick" is a constant of Employee (Nick: Employee). The most basic Description Logic Language is $\mathcal{ALC}$ (Attributive Language with Complements) with syntax:

$$\mathcal{ALC} ::= \ \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C \ .$$

The following table represents the formal semantics of $\mathcal{ALC}$ and their FOL representation [13]..

Syntax	Formal Semantics	FOL semantics
A :	$A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$	$F_A(x)$
$C \sqcap D$:	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$F_C(x) \wedge F_D(x)$
$C \sqcup D$:	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$F_C(x) \vee F_D(x)$
$\neg C$:	$\triangle^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$\neg F_C(x)$
$\forall R.C$:	$\{a \in \triangle^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	$\forall z.F_R(x,z) \rightarrow F_c(z)$
$\exists R.C$:	$\{a \in \triangle^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\} \wedge b \in C^{\mathcal{I}}\}$	$\exists z.F_R(x,z) \wedge F_c(z)$

Fig. 1. Semantics of $\mathcal{ALC}$.

Let us note that an interpretation **I** (shown in Figure 1), is a model of a knowledge base Σ, if and only if every axiom of Σ, is satisfied by **I**. Σ logically implies A ⊑ C (written Σ ⊨ A ⊑ C) if $A^I \subseteq C^I$ for every model of Σ: we say that A is subsumed by C in Σ [3].

Description logics can be extended in order to have a temporal basis. We can have either point-based temporal description logics ($\mathcal{ALCT}$ [13], $\mathcal{CQIus}$ [12] , $\mathcal{ALCQIT}$ [4], or interval based temporal description logics [13]. A way of adding tense logic (point based) to Description Logics was introduced by [12], with $\mathcal{CQIus}$, and extended by [4], with $\mathcal{ALCQIT}$. In the aforementioned extensions the temporal operators $\mathcal{U}$(until) and $\mathcal{S}$ (Since), along with the temporal operators $\Diamond^+$, $\Diamond^-$, $\Box^+$, $\Box^-$ were introduced, in order to enhance Description Logics with a temporal dimension. The meaning of those operators (also shown in Figure 2) is:

$\Diamond^+$ (Sometime in the future) $\Diamond^+ C = \top \mathcal{U} C$

$\Diamond^-$ (Sometime in the past) $\Diamond^- C = \top \mathcal{S} C$

$\Box^+$ (Always in the future)

$\Box^-$ (Always in the past)

C, D	$\rightarrow$	$C \mathcal{U} D \mid$	$(C \texttt{ until } D)$ (until)
		$C \mathcal{S} D \mid$	$(C \texttt{ since } D)$ (since)
		$\Diamond^+ C \mid$	$(\texttt{somefut } C)$ (future existential)
		$\Diamond^- C \mid$	$(\texttt{somepast } C)$ (past existential)
		$\Box^+ C \mid$	$(\texttt{allfut } C)$ (future universal)
		$\Box^- C$	$(\texttt{allpast } C)$ (past universal)

Fig. 2. Temporal Extension of $\mathcal{ALCQI}$

A further extension of the languages $\mathcal{ALCT}$, $\mathcal{ALCQIT}$ and $\mathcal{CQIus}$ was made by [9], with the language $\mathcal{ALC}_{\mu\bullet\circ}$ that defines the new operators ●(next time), ○(previous time), and fixpoint concept expressions like $\mu A.C$. A language describing Interval Based Description Logics is $\mathcal{TDL}$ presented by Lutz [10]. Also a powerful temporal description language named $\mathcal{DLRus}$ is described by [6]. $\mathcal{DLRus}$ variants accomplish EXPTIME-complete reasoning, and EXPSPACE-complete satisfiability and logical implication [6].

Artale and Franconi have also presented a language able to describe both non-temporal feature logic and interval temporal networks is $\mathcal{TL}$-$\mathcal{F}$ [3]. The basic types of this language are concepts, individuals, temporal variables and intervals. Concepts can be specified to hold at a certain temporal variable (or interval). In this way, actions (resp. individual actions) can be expressed in a uniform way by temporally related concepts (resp. individuals) [3]. The most common notation used for Temporal Interval Relations, used in Interval Based Temporal Description Logics, is the one originally presented by Allen in [1],[5], and also shown in Figure 3.

Concepts expressions are denoted by C, D are built out of atomic concepts denoted by A. Atomic features are denoted by f, whereas atomic parametric features are denoted by ⋆g. Parametric features [3], (e.g. ⋆Employee in ⋆Employee: Misdemeanor) plays the role of formal parameter of the action mapping any individual action of type Misdemeanor independently from time. Temporal variables are denoted by X, Y. When writing C@X it means that concept C holds at time interval denoted by variable X. There is also the special temporal variable **now (or #)** which is used as the reference temporal variable of the action (concept). Now variable can be omitted when possible. For example, the expression Illegal ⊑ Suspended is the same with Illegal@now ⊑ Suspended@now.

Relation	Abbr.	Inverse
before(i, j)	b	a
meets(i, j)	m	mi
overlaps(i, j)	o	oi
starts(i, j)	s	si
during(i, j)	d	di
finishes(i, j)	f	fi

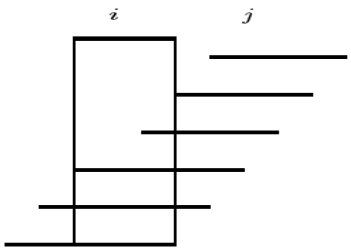


Fig. 3. Allen's interval relationships.

<concept> ::= <atomic-concept>
| (and <concept>$^+$)
| (all <role> <concept>)
| (atleast *min* <role>)
| (atmost *max* <role>)
| (at <interval> <concept>)
| (sometime (<interval-variable>$^+$) <time-net>.<concept>)
| (alltime (<interval-variable>$^+$) <time-net>.<concept>)
<atomic-concept> ::= *symbol*
<role> ::= <atomic-role>
| (and <role>$^+$)
| (domain <concept>)
| (range <concept>)
| (at <interval> <role>)
| (sometime (<interval-variable>$^+$) <time-net>.<role>)
| (alltime (<interval-variable>$^+$) <time-net>.<role>)
<atomic-role> ::= *symbol*
<time-net> ::= <time-constraint>
| (and <time-constraint>$^+$)
<time-constraint> ::= (<interval-relation> <interval> <interval>)
| (<comparison> <interval> <duration-constant>)
| (<granularity> <interval>)
<interval-relation> ::= equal | meets | met-by | after | before
| overlaps | overlapped-by | starts | started-by
| finishes | finished-by | during | contains
| (or <interval-relation>$^+$)
<comparison> ::= $< \mid \leqslant \mid = \mid \geqslant \mid >$
<granularity> ::= *sec* | *min* | *hour* | ...
<interval> ::= <interval-variable> | <interval-constant> | *now*
<interval-variable> ::= *symbol*
<interval-constant> ::= *symbol*
<duration-constant> ::= *symbol*

Fig. 4. Syntax rules of Schmiedel's Temporal Extension proposal

As we have mentioned before in the following section we combine features from the language $\mathcal{TL}$-$\mathcal{F}$ presented by [3], as well as features of the syntax rules from the Schmiedel proposal [2] (also shown in figure 4), in order to represent actions, integrity constraints and non persistent effects, in Interval Based Temporal Description Logics. Extensions have been made, in cases where the existing Temporal Description Logic Languages, were not specific enough to express actions, integrity constraints and time restrictions, with clarity as shown in the following example of Section 3.

3 Dealing with the ramification problem with algorithms expressed in Temporal Description Logics

A solution to the ramification problem with the use of Temporal Description Logics is presented in this section. Consider the following example from [11]): If a public employee commits a misdemeanor, then for the next five months s/he is considered illegal, except if s/he receives a pardon. When a public employee is illegal, then s/he must be suspended and cannot take promotion, for the entire time interval over which s/he is considered illegal. Also when a public employee is suspended s/he cannot take her/his salary until s/he stops being suspended. Each public employee is evaluated for her/his work. If s/he receives a bad grade, then s/he is assumed to be a bad employee and s/he cannot take promotion until s/he receives a good grade. If s/he receives a good grade, then s/he is assumed to be a good employee and s/he takes a bonus if s/he is not suspended. Also, assume that a public worker is not illegal, if there does not exist information that proves s/he is illegal, and is not suspended if there does not exist information that proves s/he is suspended and takes his/her salary, if there does not exist information that proves the opposite. This helps us define the default axioms. As we observe we have four actions *misdemeanor, take_pardon, good_grade, bad-grade* and seven fluents: good_*employee, bad_employee, illegal, take_salary, take_bonus, take_promotion, suspended*. The direct effects of the four actions are expressed in propositional form by the following constraints:

FOL Representation of the Integrity Constraints as presented by Papadakis and Plexousakis [11]:

occur(misdemeanor(p), t) ⊃ illegal(p,5m) **(1)**
occur(take_pardon(p), t) ⊃ ¬illegal(p, ∞) **(2)**
occur(bad_grade(p), t) ⊃ ¬good_employee(p, ∞) **(3)**
occur(good_grade(p),t) ⊃ good_employee(p, ∞) **(4)**

t is a temporal variable and occur(misdemeanor(p), t) means that the action misdemeanor(t) takes place at time t. The following integrity constraints describe the indirect effects of the four aforementioned actions.

illegal(p, t1) ⊃ suspended(p, t1) **(5)**
illegal(p,t1) ⊃ ¬ take_promotion(p, t1) **(6)**
suspended(p, t1) ⊃ ¬ take_salary(p, t1) **(7)**
¬ good_employee(p, t1) ⊃ ¬ take_promotion(p, t1) **(8)**
¬ suspended(p, t1) ∧ good_employee(p, t2) ⊃ take_bonus(p, min(t1,t2)) **(9)**
¬ good_employee(p,t1) ⊃ ¬ take_bonus(p, t1) **(10)**
¬ suspended(p, t1) ⊃ take_salary(p, t1) **(11)**

Temporal Description Logics Representation of the Integrity Constaints:

$\Diamond^+$ (x) (**starts now** x) (month x) (= x 5)(⋆Employee: Misdemeanor ⊑ ⋆Employee: Illegal@x) **(1)**
$\Diamond^+$ (x) (**starts now** x) (⋆Employee: Take_pardon ⊑ ⋆Employee: ¬ Illegal@x) **(2)**

$\Diamond^+$ (x) (**starts now** x) (⋆Employee: Bad_grade ⊑ ⋆Employee:¬Good_employee@x) **(3)**

$\Diamond^+$ (x) (**starts now** x) (⋆Employee: Good_grade ⊑ ⋆Employee: Good_employee@x) **(4)**

Also we have got the following **integrity constraints** describing the **indirect effects** of the aforementioned actions.

⋆Employee:Illegal ⊑ ⋆Employee:Suspended **(5)**

⋆Employee:Illegal ⊑ ⋆Employee: ¬Take_promotion **(6)**

⋆Employee:Suspended ⊑ ⋆Employee: ¬Take_salary **(7)**

⋆Employee: ¬ Good_employee ⊑ ⋆Employee: ¬Take_promotion **(8)**

$\Diamond^+$ (x y z)(**starts now** x)(**starts now** y)(= z min(x, y)) (⋆Employee: ¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z) **(9)**

⋆Employee:¬Good_employee ⊑ ⋆Employee:¬Take_bonus **(10)**

⋆Employee:¬Suspended ⊑ ⋆Employee: Take_salary **(11)**

In Temporal Settings we need to describe the direct and indirect effects of an action, not only in the immediately resulting next situation, but possibly for many future situations. In the above example, the action Misdemeanor has the indirect effect that the public worker is in suspension in the next five months. In these five months the action Good_grade may occur, but even if that happens, the employee still cannot take promotion. This means that the world changes situations while the direct and indirect effects of some actions still hold. In the above example the dynamic axioms are the (1) – (4), while the static axioms are the rest (5) – (11). We have the following **default axioms**:

$\Diamond^+$ (x y) (**starts** now x)(**starts** now y) (= x 0) (⋆Employee: Illegal@x ⊓ ⋆Employee:¬Illegal@x ⊑ ⋆Employee: ¬Illegal@y)

$\Diamond^+$ (x y) (**starts** now x)(**starts** now y) (= x 0) (⋆Employee: Take_salary@x ⊓ ⋆Employee: ¬Take_salary@x ⊑ ⋆Employee: Take_salary@y)

$\Diamond^+$ (x y) (**starts** now x)(**starts** now y) (= x 0) (⋆Employee: Suspended@x ⊓ ⋆Employee: ¬Suspended@x ⊑ ⋆Employee: ¬Suspended@y)

a. Algorithms for the Production of Static Rules

The static rules can be used to deduct the indirect effects of the execution of each action. The indirect effects exist due to the presence of integrity constraints. Therefore, it is possible to produce the static rules, from the integrity constraints. The following algorithm describes the steps needed for the aforementioned production.

1. Transform each integrity constraint in its CNF (conjunctive) form. Now each integrity constraint has the form $C_1 \sqcap C_2 \sqcap C_3 \ldots \sqcap C_n$
2. For each i from 1 to n do

Assume $C_i = F_1 \sqcup F_2 \ldots \sqcup F_m$

For each j from 1 to m do

For each k from 1 to m and $k \neq j$ do

if $(F_j, F_k) \in I$ then

$R = R \sqcup (\neg F_j$ causes F_k if $\prod F_l)$, l=1,…,m, $l \neq j$, k

3. For each fluent F_k the rules have the following form:

$\prod F_i$ causes F_k if Φ

$\prod F_i$' causes $\neg F_k$ if Φ'

We change the static rules from the form:

$G \sqsubseteq F_k$

$K \sqsubseteq \neg F_k$

to the form:

$G' \sqsubseteq F_k$

$K' \sqsubseteq \neg F_k$

where

$G' = G \sqcup (\prod F_i \sqcap \Phi)$

$K' = K \sqcup (\prod F_i' \sqcap \Phi')$

4. We replace each rule $G_p \sqsubseteq F_p$ with $\Diamond^+ (x)$(**starts** now x)(G_p@x $\sqsubseteq$ F_p@x)

Now we apply the algorithm to the previous example. First we have to produce the set *I*. In order to do that, we make use of an algorithm presented by Papadakis and Plexousakis [11]. The algorithm is however described with Description Logics this time.

1. For each fluent $F \in G_f$, $F' \in K_f$, where $G_f \sqsubseteq K_f$ is a specified integrity constraint add the pair $(F, F') \in I$.
2. For each $F \in G_f$, $F' \in K_f$, where $G_f \equiv K_f$ is a specified constraint do
   - If F can change its truth values as the direct effect of an action, then add (F,F') in *I*. If F' can change its truth value as a direct effect of an action then add (F',F) in *I*.

All the integrity constraints (IC) have the form $A \sqsubseteq B$. We have that:

(Illegal, Suspended) ∈ I (from IC 5)

(Illegal, ¬Take_promotion) ∈ I (from IC 6)

(Suspended, ¬Take_salary) ∈ I (from IC 7)

(¬Good_employee, ¬Take_promotion) ∈ I (from IC 8)

(¬Suspended, Take_bonus) ∈ I (from IC 9)

(Good_employee, Take_bonus) ∈ I (from IC 9)

(¬Good_employee, ¬ Take_bonus) ∈ I (from IC 10)

(¬Suspended, Take_salary) ∈ I (from IC 11)

The transformation of integrity constraints in conjunctive normal form (step 1) yields:

⋆Employee: ¬Illegal ⊔ ⋆Employee: Suspended **(5)**

⋆Employee: ¬Illegal ⊔ ⋆Employee: ¬Take_promotion **(6)**
⋆Employee: ¬Suspended ⊔ ⋆Employee: ¬Take_salary **(7)**
⋆Employee: Good_employee ⊔ ⋆Employee: ¬ Take_promotion **(8)**
$\Diamond^+$ (x y z) (**starts** now x)(**starts** now y)(= z min(x,y)) ⋆Employee: Suspended@x ⊔ ⋆Employee: ¬Good_employee@y ⊔ ⋆Employee: Take_bonus@z **(9)**
⋆Employee: Good_employee ⊔ ⋆Employee: ¬Take_bonus **(10)**
⋆Employee: Suspended ⊔ ⋆Employee: Take_salary **(11)**

In step 2 we estimate all causal relationships omitting the atomic parametric feature "⋆Employee" for better readability.

R = {Illegal *causes* Suspended if ⊤, Illegal *causes* ¬Take_promotion if ⊤,
Suspended *causes* ¬Take_salary if ⊤, ¬Good_employee *causes* ¬Take_promotion if ⊤,
Good_employee *causes* Take_bonus if ¬Suspended,
¬Suspended causes Take_bonus if Good_employee,
¬Good_employee *causes* ¬Take_bonus if ⊤, ¬Suspended *causes* Take_salary if ⊤}.

In the following step (step 3), we construct the fluent formulas, which make each fluent true. In case that there are more than one causal relationships affecting the same fluent, we integrate them in this step. Take for example the second and fourth causal relationships from step 2.

R = {Illegal ⊑ Suspended,
Illegal ⊔ ¬Good_employee ⊑ ¬Take_promotion,
Suspended ⊑ ¬Take_salary,
¬Suspended ⊓ Good_employee ⊑ Take_bonus,
¬Good_employee ⊑ ¬Take_bonus,
¬Suspended ⊑ Take_salary }.

Finally in step 4 (which must be exectued, at each time point, at which the static rules are evaluated) we have the following six static rules:

R = {$\Diamond^+$ (x)(**starts** now x)(⋆Employee: Illegal@x ⊑ ⋆Employee: Suspended@x),
$\Diamond^+$ (x y z)(**starts** now x)(**starts** now y)(= z max(x,y))(⋆Employee:Illegal@x ⊔ ⋆Employee: ¬Good_employee@y ⊑ ⋆Employee: ¬Take_promotion@z),
$\Diamond^+$ (x)(**starts** now x)(⋆Employee: Suspended@x ⊑ ⋆Employee: ¬Take_salary@x),
$\Diamond^+$ (x y z)(**starts** now x)(**starts** now y)(= z min(x, y)) (⋆Employee: ¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z),
$\Diamond^+$ (x)(**starts** now x)(⋆Employee: ¬ Good_employee@x ⊑ ⋆Employee: ¬Take_bonus@x),
$\Diamond^+$ (x)(**starts** now x)(⋆Employee: ¬Suspended@x ⊑ ⋆Employee: Take_salary@x) }.

In step 4, for each static rule, we find the maximum time that its body is true. For example in the second rule, the body is true when Illegal@x ⊔ ¬Good_employee@y is true. This means that we take

the maximum of times x, y for which ¬Take_promotion is true. On the fourth rule, the body is true when ¬Suspended@x ⊓ Good_employee@y is true. This means that we must take the minimum of times x, y.

b. Algorithms for the Evaluation of Dynamic and Static Rules

In this section we present an algorithm for the evaluation of rules:

1. After the execution of one action evaluate the dynamic rule which references at this action.
2. Each time moment do:
    a. Evaluate the default axioms
    b. Repeat until no change occurs
        i. Evaluate all static rules
        ii. If a fluent $\diamondsuit^+$ (a x)(**starts** a x)(F@x) becomes true after the evaluation of a static rule, then set $\diamondsuit^+$ (a y)(**starts** a y)(= y 0)(¬F@y) (the negation is false), for the same time interval a as in the previous rule.

Consider the above example with the public worker. We have four dynamic rules (1-4) as we have described in the previous section. Also we have produced the **six static rules**:

$\diamondsuit^+$ (x)(**starts now** x)(⋆Employee: Illegal@x ⊑ ⋆Employee: Suspended@x),
$\diamondsuit^+$ (x y)(**starts now** x)(**starts now** y)(= z max(x,y))(⋆Employee: Illegal@x ⊔ ⋆Employee: ¬Good_employee@y ⊑ ⋆Employee: ¬Take_promotion@z),
$\diamondsuit^+$ (x)(**starts now** x)(⋆Employee: Suspended@x ⊑ ⋆Employee: ¬Take_salary@x),
$\diamondsuit^+$ (x y z)(**starts now** x)(**starts now** y)(= z min(x, y)) (⋆Employee: ¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z),
$\diamondsuit^+$ (x)(**starts now** x)(⋆Employee: ¬ Good_employee@x ⊑ ⋆Employee: ¬Take_bonus@x),
$\diamondsuit^+$ (x)(**starts now** x)(⋆Employee: ¬Suspended@x ⊑ ⋆Employee: Take_salary@x)

Assume now that we have a public worker (employee) ⋆E, and the initial situation is:

S_0 = {$\diamondsuit^+$ (x) (starts now x)(⋆E :¬Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: ¬Good_employee@x, ⋆E: ¬Illegal@x)}

Time starts at 0 and has the granularity of month. Assume that the following actions occur at the following time points:

⋆E: Good_grade@2
⋆E: Misdemeanor@4
⋆E: Bad_grade@6

⋆E: Good_grade@8
⋆E: Misdemeanor@10
⋆E: Take_pardon@12

At time point 2 the action Good_grade executes. According to the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (4) we have the following situation:

S'_1 = {$\Diamond^+$ (x) (starts now x) (⋆E:¬Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: Good_employee@x, ⋆E: ¬Illegal@x)}

As we observe the following static rule will be evaluated:

$\Diamond^+$ (x y z)(starts now x)(starts now y)(starts now z)(= z min(x,y))(⋆E: ¬Suspended@x ⊓ ⋆E: Good_employee@y ⊑ ⋆E: Take_bonus@z)

In the previous rule the (starts now z) could be omitted, as z can be either x or y. Before the action execution, time interval x has the maximum value [now, ∞). Therefore it is safe to assume that the minimum time interval among x and y is y, as the time interval updated in this rule is y from fluent Good_employee. Therefore in the above rule z=y.

S_1 = {$\Diamond^+$ (x) (starts now x) (⋆E: Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: Good_employee@x, ⋆E: ¬Illegal@x)}

The situation does not change until timepoint 4, when the second action (Misdemeanor) takes place and causes Illegal to become true from the next 5 time units. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule(1) we have the situation:
S_2' = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 5) (⋆E :Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: Good_employee@x, ⋆E: Illegal@y)}

As we observe the following static rules will be evaluated:

$\Diamond^+$ (x)(**starts now** x)(⋆Employee:Illegal@x ⊑ ⋆Employee:Suspended@x),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee:Suspended@x ⊑ ⋆Employee: ¬Take_salary@x).

x has duration 5 so we get:

$\Diamond^+$ (x)(**starts now** x)(= x 5)(⋆Employee:Illegal@x ⊑ ⋆Employee:Suspended@x),
$\Diamond^+$ (x)(**starts now** x)(= x 5)(⋆Employee:Suspended@x ⊑ ⋆Employee: ¬Take_salary@x),

After the evaluation of the static rules we have the situation:

S_2 = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 5) (⋆E :Take_bonus@x, ⋆E: ¬Take_salary@y, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@y, ⋆E: Good_employee@x, ⋆E: Illegal@y)}

This situation does not change until the time point 6, when the third action (Bad_grade) executes. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (3) we have the situation:

S_3' = {$\Diamond^+$ (x y) (starts now x) (starts now y)(= y 3) (⋆E :Take_bonus@x, ⋆E: ¬Take_salary@y, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@y, ⋆E:¬Good_employee@x, ⋆E: Illegal@y)}

The following static rule:

$\Diamond^+$ (x)(**starts now** x)(⋆Employee: ¬Good_employee@x ⊑ ⋆Employee: ¬Take_bonus@x), will be evaluated and the situation will become:

S_3 = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 3) (⋆E:¬Take_bonus@x, ⋆E:¬Take_salary@y, ⋆E:¬Take_promotion@x, ⋆E: Suspended@y, ⋆E: ¬Good_employee@x, ⋆E: Illegal@y)}

This situation does not change until time point 8, when the fourth action (Good_Grade) takes place. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (4), we have the situation:

S_4' = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 1) (⋆E : ¬Take_bonus@x, ⋆E:¬Take_salary@y, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@y, ⋆E: Good_employee@x, ⋆E: Illegal@y)}

No static rule is executed. Therefore the situation does not change. At time point 9 no action takes place, but the situation changes because the following default axioms hold:
$\Diamond^+$ (x y) (**starts** now x)(**starts** now y) (= x 0) (⋆Employee: Illegal@x ⊓ ⋆Employee:¬Illegal@x ⊑ ⋆Employee: ¬Illegal@y)

$\Diamond^+$ (x y) (**starts** now x)(**starts** now y)(= x 0) (⋆Employee: Take_salary@x ⊓ ⋆Employee: ¬Take_salary@x ⊑ ⋆Employee: Take_salary@y)

$\Diamond^+$ (x y) (**starts** now x)(**starts** now y) (= x 0) (⋆Employee: Suspended@x ⊓ ⋆Employee: ¬Suspended@x ⊑ ⋆Employee: ¬Suspended@y)

The situation is:

S_5' = {$\Diamond^+$ (x) (starts now x) (⋆E: ¬Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E:¬Suspended@x, ⋆E: Good_employee@x, ⋆E:¬Illegal@x)}

Now the static rule:

$\Diamond^+$ (x y z)(**starts** now x)(**starts** now y)(= z min(x, y)) (⋆Employee:¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z), is executed with x=y and therefore z=x=y. After the evaluation of the rule we have:

S_5 = {$\Diamond^+$ (x) (starts now x) (⋆E: Take_bonus@x, ⋆E:Take_salary@x, ⋆E:¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: Good_employee@x, ⋆E:¬Illegal@x)}

At time point 10 the action Misdemeanor executes, resulting the situation:

S_6' = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 5)(⋆E : Take_bonus@x, ⋆E:Take_salary@x, ⋆E:¬Take_promotion@x, ⋆E:¬Suspended@x, ⋆E: Good_employee@x, ⋆E: Illegal@y)}

$\Diamond^+$ (x)(**starts** now x)(= x 5)(⋆Employee:Illegal@x ⊑ ⋆Employee:Suspended@x),
$\Diamond^+$ (x)(**starts** now x) (= x 5)(⋆Employee:Suspended@x ⊑ ⋆Employee: ¬Take_salary@x).

S_6 = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 5)(⋆E: Take_bonus@x, ⋆E: ¬Take_salary@y, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@y, ⋆E: Good_employee@x, ⋆E: Illegal@y)}

The last action (Take_pardon) occurs at time point 12. The new situation is

S_7' = {$\Diamond^+$ (x y) (starts now x)(starts now y)(= y 3)(⋆E : Take_bonus@x, ⋆E: ¬Take_salary@y, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@y, ⋆E: Good_employee@x, ⋆E:¬Illegal@x)}

Finally the situation changes again at time point 15, because the following default axioms hold:

$\Diamond^+$ (x y) (**starts now** x)(**starts now** y)(= x 0) (⋆Employee: Take_salary@x ⊓ ⋆Employee: ¬Take_salary@x ⊑ ⋆Employee: Take_salary@y)

$\Diamond^+$ (x y) (**starts now** x)(**starts now** y) (= x 0) (⋆Employee: Suspended@x ⊓ ⋆Employee: ¬Suspended@x ⊑ ⋆Employee: ¬Suspended@y)

Now the situation is:

S_8 = {$\Diamond^+$(x) (starts now x) (⋆E : Take_bonus@x, ⋆E: Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: Good_employee@x, ⋆E: ¬Illegal@x)}

This is the end of execution. As we observe from the set R, for each pair (F, ¬F) it holds that G_F ⊓ K_F ≡ FALSE, when G_F ⊑ F and K_F ⊑ ¬F. More specifically the set of static rules is:

R = {$\Diamond^+$ (x)(**starts now** x)(⋆Employee: Illegal@x ⊑ ⋆Employee: Suspended@x),
$\Diamond^+$ (x y z)(**starts now** x)(**starts now** y)(= z max(x,y))(⋆Employee:Illegal@x ⊔ ⋆Employee: ¬Good_employee@y ⊑ ⋆Employee: ¬Take_promotion@z),

$\Diamond^+$ (x)(**starts now** x)(⋆Employee: Suspended@x ⊑ ⋆Employee: ¬Take_salary@x),
$\Diamond^+$ (x y z)(**starts now** x)(**starts now** y)(= z min(x, y)) (⋆Employee:¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee: ¬ Good_employee@x ⊑ ⋆Employee:¬Take_bonus@x),
$\Diamond^+$ (x)(**starts now** x)(⊥ ⊑ ⋆Employee:¬Suspended @x),
$\Diamond^+$ (x)(**starts now** x)(⊥ ⊑ ⋆Employee: Take_promotion@x)
$\Diamond^+$ (x)(**starts now** x)(⊥ ⊑ ⋆Employee: Illegal@x)
$\Diamond^+$ (x)(**starts now** x)(⊥ ⊑ ⋆Employee:¬Illegal@x)
}, where ⊥ is the symbol for FALSE.

As we observe, for the fluent that there is not a static rule, we add the rule FALSE ⊑ F, because they cannot become true by static rules, but only by dynamic rules (this means that the truth value changes only as the direct effect of some action). Now we have:

$\Diamond^+$ (x y)(**starts now** x)(**starts now** y)((⋆Employee: ¬Suspended@x ⊓ ⋆Employee: Good_employee@y) ⊓ ⋆Employee:¬Good_Employee@x) for (Take_bonus, ¬Take_bonus)

$\Diamond^+$ (x y)(**starts now** x) (⋆Employee: Suspended@x ⊓ ⋆Employee: ¬Suspended@x) for (Take_salary, ¬Take_Salary)

$\Diamond^+$ (x)(**starts now** x)(⋆Employee:Illegal@x ⊓ ⊥) for (Suspended, ¬Suspended)

$\Diamond^+$ (x y)(**starts now** x)(**starts now** y)((⋆Employee:Illegal@x ⊔ ⋆Employee: ¬Good_employee@y) ⊓ ⊥) for (Take_promotion, ¬Take_promotion)

⊥ ⊓ ⊥ for (Illegal, ¬Illegal)

This assumption $G_f \sqcap K_f \equiv \bot$ is very important in order to ensure that, always after the execution of action there is a consistent situation. Now we show with an example, that if this assumption does not hold, the situation is not consistent after the execution of some sequence of actions.

Consider the above example with the public worker and assume that there is another integrity constraint specifying that when a public worker is Good_employee, then s/he takes promotion. Now the set of static rules is:

R = {$\Diamond^+$ (x)(**starts now** x)(⋆Employee:Illegal@x ⊑ ⋆Employee:Suspended@x),
$\Diamond^+$ (x y z)(**starts now** x)(**starts now** y)(= z max(x,y))(⋆Employee:Illegal@x ⊔ ⋆Employee: ¬Good_employee@y ⊑ ⋆Employee: ¬Take_promotion@z),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee:Suspended@x ⊑ ⋆Employee: ¬Take_salary@x),
$\Diamond^+$ (x y z)(**starts now** x)(**starts now** y)(= z min(x, y)) (⋆Employee: ¬Suspended@x ⊓ ⋆Employee: Good_employee@y ⊑ ⋆Employee: Take_bonus@z),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee: ¬ Good_employee@x ⊑ ⋆Employee: ¬Take_bonus@x),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee: ¬Suspended@x ⊑ ⋆Employee: Take_salary@x),
$\Diamond^+$ (x)(**starts now** x)(⋆Employee: Good_employee@x ⊑ ⋆Employee: Take_promotion@x)

}.

As we observe for the pair (Take_promotion, ¬Take_promotion), the above assumption does not hold, because

$\diamondsuit^+$ (x)(**starts now** x)(starts now y)(Good_employee@x ⊓ (Illegal@x ⊔ ¬Good_employee@y)) can be true when Good_employee ⊓ Illegal holds.

Assume now that we have a public worker ⋆E and the initial situation is:

S_0 = {$\diamondsuit^+$ (x) (starts now x) (⋆E:¬Take_bonus@x, ⋆E:Take_salary@x, ⋆E:¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E:¬Good_employee@x, ⋆E:¬Illegal@x)}

As we have mentioned before, x has the maximum value possible for a future time interval which is [now, ∞). Assume that the following actions occur at the following time points, assuming time starts at 0 and time granularity is that of months.

Misdemeanor@4
Good_grade@6

At time point 4 after the execution of the action Misdemeanor we have the situation:

S_1' = {$\diamondsuit^+$ (x) (starts now x) (⋆E:¬Take_bonus@x, ⋆E:Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: ¬Suspended@x, ⋆E: ¬Good_employee@x, ⋆E: Illegal@x)}

After the evaluation of the static rules we have:

S_1 = {$\diamondsuit^+$ (x) (starts now x) (⋆E:¬Take_bonus@x, ⋆E:¬Take_salary@x, ⋆E: ¬Take_promotion@x, ⋆E: Suspended@x, ⋆E: ¬Good_employee@x, ⋆E: Illegal@x)}

At time point 6, after the execution of the action Good_grade we have the situation:

S_2' = {$\diamondsuit^+$ (x) (starts now x) (⋆E:¬Take_bonus@x, ⋆E:¬Take_salary@x, ⋆E:¬Take_promotion@x, ⋆E: Suspended@x, ⋆E: Good_employee@x, ⋆E: Illegal@x)}

Now the static rule $\diamondsuit^+$(x)(**starts now** x)(⋆Employee: Good_employee@x ⊑ ⋆Employee: Take_promotion@x) must be evaluated, and after that Take_promotion@x must hold. But if Take_promotion@x holds, then we must examine if the static rule $\diamondsuit^+$(x y z)(**starts now** x)(**starts now** y)(= z max(x,y))(⋆Employee: Illegal@x ⊔ ⋆Employee:¬Good_employee@y ⊑ ⋆Employee:¬Take_promotion@z), must be evaluated. We observe that we must evaluate this static rule as well. As we see, those two static rules will be evaluated one after the other for ever (infinitely). This means that the situation is not consistent. This happened because there is a mistake in the integrity constraints, and therefore the above assumption does not hold. The algorithm can run without the above assumption, but we must determine the preconditions of each action, in order to avoid the above problem. We have proved the following theorems:

Theorem 1: Each time unit, the algorithm is terminated at a finite number of steps.
Theorem 2: *The above algorithm always returns a legal situation.*

4 Conclusion

In this paper we have presented the basics of Description Logics, as well as several approaches to Temporal Description Logics found in Literature. We also presented a Temporal Description Logics representation, used in a thorough example to come up with a solution to the ramification problem in Temporal Settings. In order to accomplish that, we present algorithms that utilize Integrity constraints along with static and dynamic rules, all expressed in Temporal Description Logics. In this particular example time intervals are not only variables but can have enumerated values, in contrast with most examples found in Literature.

As we showed it is possible to deal with the ramification problem with rules and algorithms expressed in whole in Temporal Description Logics. Our implementation can also work with actions taking place in the past, by applying the same algorithms to evaluate static and dynamic rules and to come up with a consistent situation. The Temporal Description Logics representation and algorithms we presented, could also work with non instant actions (actions with duration).

References

1. Allen, J. F. & Ferguson, G.: Actions and Events in Interval Temporal Logic (1994)
2. Artale, A. & Franconi, E.: A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence* **30** (1-4), pp 171--210 (2000)
3. Artale, A. & Franconi, E.: A Temporal Description Logic for Reasoning about Actions and Plans. *J. Artif. Int. Res.* **9** (1), pp 463--506 (1998)
4. Artale, A. & Franconi, E.: Introducing Temporal Description Logics. In TIME '99: Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (pp. 2). IEEE Computer Society. (ISBN: 0-7695-0173-7.) (1999)
5. Artale, A. & Franconi, E.: Temporal Description Logics. In Handbook of Time and Temporal Reasoning in Artificial Intelligence. MIT Press (2000)
6. Artale, A., Franconi, E., C, M. M., Frank, W. & Zakharyaschev, M.: The DLR_US Temporal Description Logic. In Handbook of Time and Temporal Reasoning in Artificial Intelligence (pp. 96-105). MIT Press (2001)
7. Baader, F. & Nutt, W.: Basic Description Logics. , pp 43--95 (2003)
8. Calvanese, D., Giacomo, G. D., Nardi, D. & Lenzerini, M.: Reasoning in Expressive Description Logics. , pp 1581--1634 (2001)
9. Franconi, E. & Toman, D.: Fixpoint Extensions of Temporal Description Logics. In Description Logics (2003)
10. Lutz, C.: Interval-based Temporal Reasoning with General TBoxes. In *IJCAI* pp. 89--96 (2001)
11. Papadakis, N. & Plexousakis, D.: Actions with Duration and Constraints: the Ramification Problem in Temporal Databases. In *ICTAI* pp. 83--90 (2002)
12. Wolter, F. & Zakharyaschev, M.: Temporalizing Description Logics. In *In Proceedings of FroCoS'98* pp. 104--109 (1999)
13. Zhang, J.: Description Logics and Time (2006)

Scalability via Parallelization of OWL Reasoning

Thorsten Liebig, Andreas Steigmiller, and Olaf Noppens

Institute for Artificial Intelligence, Ulm University
89069 Ulm, Germany
firstname.lastname@uni-ulm.de

Abstract. Practical scalability of reasoning is an important premise for the adoption of semantic technologies in a real-world setting. Many highly effective optimizations for reasoning with expressive OWL ontologies have been invented and implemented over the last decades. This paper describes our approach for concurrent computation of the nondeterministic choices inherent to the OWL tableau reasoning procedure for $\mathcal{SHIQ}$. We present the architecture of our parallel reasoner and briefly discuss our prototypical implementation as well as future work.

Key words: OWL, Reasoning, Parallelization

1 Motivation

Tableaux-based algorithms have shown to be an adequate method in order to implement OWL reasoning services for many practical use-cases of moderate size. However, scalability of OWL reasoning is still an actual challenge of DL research. Recent optimizations have shown significant increase in speed for answering queries with respect to large volumes of individual data under specific conditions. For instance, the KAON2 [1] system achieves excellent performance for reasoning with large volumes of individuals by a clever transformation of OWL into disjunctive Datalog unless there are cardinality restrictions. A recently proposed variant of the tableau algorithm [2] has shown some speed-ups for at least certain kind of ontologies within the HermiT reasoning system. SHEER, a different, sound but incomplete approach [3] tries to reason with a condensed version of the data but is not applicable in the presence of nominals. The CB system implements a consequence driven approach [4] which is theoretically optimal at least for the Horn-fragment of $\mathcal{SHIQ}$. The bottom line is that almost all optimizations typically do come with some restriction in expressivity. This, however, adds another critical dimension to developers of semantic applications in that they need to hit the right language fragment which hopefully suits their needs but also comes with a powerful reasoning engine.

On the other hand, modern CPU's typically pool more than one processing unit on a single chip. Recent consumer desktops even come with two quad-core processors. However, research into reasoning engines which distribute their work load in such a setting just has started ([5, 6]). Clearly, parallel computation can only reduce processing time by a factor which is determined by the available

processing units but has the potential of being applicable without any restriction especially to the most "costly" cases.

On a high level there are at least two different approaches for parallelizing reasoning:

Reasoner level. Refers to the approach where the system runs independent instances of the reasoner procedure to solve some service task. For instance, computation of the class hierarchy can be delegated to a set of reasoner instances each of which check consecutively for subsumption of two classes.

Proof level. Aims at parallelizing the reasoning procedure itself by concurrent computation of inherently independent proof steps.

Our approach follows the proof level strategy because of its sophisticated tuning and optimization options. The reasoner level approach, by contrast, is a naive kind of parallelization whose synchronization interval is more or less unpredictable and therefore far from optimal. For instance, the efficient computation of the concept hierarchy should exploit previous subsumption results. In the worst case the reasoner level approach has to compute some tests multiple times due to inherent poor synchronization possibilities.

This paper describes how to parallelize the well-known tableau algorithm used within reasoning systems such as RacerPro, FaCT++, or Pellet. Parallelizing the nondeterministic choices within the standard DL tableau procedure has several advantages. First of all, nondeterminism is inherent to the tableau algorithm due to logical operators such as disjunction, at-most, or qualified cardinality restrictions. The generated alternatives from these expressions are completely independent of each other and can be computed concurrently. In case of a positive result the other sibling threads can be aborted. The parallel computation of nondeterministic alternatives also makes the algorithm less dependent on heuristics which typically choose the next alternative to process. A bad guess within a sequential algorithm inevitably will lead to a performance penalty. A parallel approach has the advantage of having better odds with respect to at least one good guess.

In the following we present our framework for distributed tableaux proofs, describe the status of our implementation and comment on first result as well as discuss future work.

2 An Approach for Parallelizing DL Tableaux Proofs

Our approach aims at parallelizing the sequential algorithm proposed in [7] for $\mathcal{ALCNH}_{R^+}$ ABoxes with GCIs, enhanced with inverse roles and qualified cardinality restrictions – referred to as $\mathcal{SHIQ}$ – and extends our previous work dealing with a parallel $\mathcal{SHN}$ reasoning system [8]

Every standard reasoning task can be reduced to a corresponding ABox unsatisfiability problem. A tableau prover will then try to create a model for this ABox. This is done by building up a tree (the tableau) of generic individuals

a_i (the nodes of the tableau) by applying tableaux expansion rules. Tableaux expansion rules either decompose concept expressions, add new individuals or merge existing individuals.

2.1 Non-Determinism within $\mathcal{SHIQ}$

The most obvious starting point for parallel proof processing are the nondeterministic tableaux rules. Nondeterministic branching yields to multiple alternatives, which can be seen as different possible ABoxes to continue reasoning with. Within $\mathcal{SHIQ}$ there are three inherent nondeterministic rules:

The disjunction rule. If for an individual a the assertion $a : C \sqcup D$ is in the ABox, then there are two possible ABoxes to continue with: either with C or with D.

The number restriction merge rule. The at-most restriction results in nondeterminism, if there are m r-successors in an ABox as well as an at-most restriction $(\leq n\ r)$ and it holds that $m > n$. In such a situation the m existing successors need to be merged to at most n r-successors. In the worst case this requires to check for all possible m on n partitions.

The choose rule. For an ABox with the qualified number restriction $(\leq n\ r.C)$ the algorithm has to add either C or $\neg C$ to any r-successor.

As there are no dependencies between the alternatives generated by the rules above. Consequently they can be evaluated within parallel threads independently.

2.2 Work Pool Architecture

In order to enable parallelism without recursively creating an overwhelming number of threads, we decided to adopt a *work pool* design as shown in Figure 1. A *work unit* with a fixed number of *work executors* is generated at the start of the tableau proof. This parametrizable number typically will be equal to or less than the number of available processing units. These executors have synchronized read access to a common queue of jobs (i.e. the ABoxes to evaluate). On start, the tableaux root node (the original query ABox) is send from a superordinate *work distributor* to a work unit (cf. step A of Fig. 1).

The unit's *work controller* will add this work package to the units work queue (step 1). The controller then starts it's executors and one of them will fetch the initial job (step 2). During processing the executors have concurrent read as well as synchronized write access to two global caches (step 3). In case of a nondeterministic rule application an executor will generate the necessary alternative ABoxes by extending the preceding ABox (step 4). This work package is prioritized and submitted to the corresponding work queue. The next available executor will fetch the most prioritized work package from the pool. The executors also report any proof relevant information such as satisfiability results to the managing work controller, which then will control other executors when appropriate (step 4).

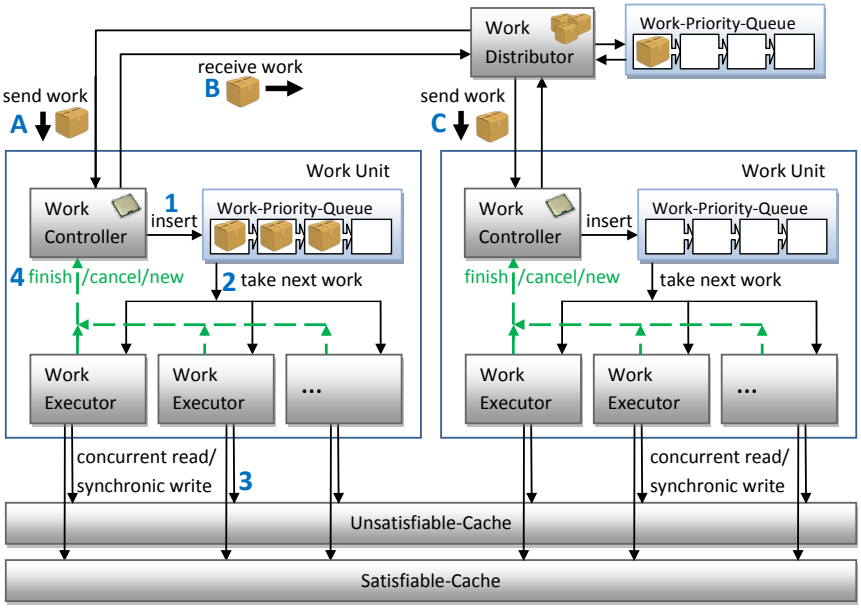


Fig. 1. Component interaction within work pool design of parallel reasoner.

The work controller itself gives notice to the work distributor in case

i) an ABox that represents a complete tableau is found, or
ii) no satisfiable alternative was found and there are no alternatives left to process.

The work distributor of our system architecture is designed to be able to coordinates more than one work unit. For instance, in case of an empty work queue of one unit the distributor will level the queues of its units such that there is no idle unit as long as there are work packages to process.

So far the design is tailored to a SMP (symmetric multi processor) architecture, where all processing cores have access to one main memory. However, our approach also allows for distribution over many computing systems. In such a setting multiple work distributors can coordinate their work between each other.

An important decision in this design is the choice of the units work pool organization. The commonly used queue is unsuitable in this setting as it promotes a breadth-first style evaluation order. Thus, ABoxes which were created earlier (generated by fewer applications of nondeterministic rules) are preferred, and the discovery of complete ABoxes is delayed. The usage of a stack would not reliably lead to a depth-first oriented processing order either, because several executor share one pool and push new work package when the occur.

We therefore have chosen to use a priority queue in order to be able to explicitly influence the processing order. We use a simple heuristic to control the processing order:

- The priority of the original ABox is set to 0.
- ABoxes generated from an ABox with priority n are given the priority $n+1$.

This allows for a controlled depth-first oriented processing order. More sophisticated heuristics or even some sort of A*-algorithm would also be possible. For example, FaCT++ also utilizes a priority queue for its ToDo list [9], weighting tableaux rules with different priorities.

2.3 Implementation Status and Future Work

The architecture as described above has been implemented in C++ utilizing the Qt libraries[1]. Qt allows for platform independent development and supports parallel processing via dedicated thread libraries. So far we only have a reasoner core in the sense that there is a $\mathcal{SHIQ}$ proof procedure only. However, we plan to build a complete reasoning system by adding the well-known pre-processing mechanisms (such as GCI absorption) and inference services (such as taxonomy computation, basic asks, etc.). Initial benchmarks also have revealed encouraging results which are planned to be published very soon.

References

1. Hustadt, U., Motik, B., Sattler, U.: Reasoning in Description Logics by a Reduction to Disjunctive Datalog. Journal of Automated Reasoning (2007) To appear.
2. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research **36** (2009) 165–228
3. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The Summary Abox: Cutting Ontologies Down to Size. In: Proceedings of the International Semantic Web Conference (ISWC 2006), Athens, GA, USA (2006) 343–356
4. Kazakov, Y.: Consequence-Driven Reasoning for Horn $\mathcal{SHIQ}$ Ontologies. In: Proceedings of the 21st International Conference on Artificial Intelligence (IJCAI 2009). (July 11-17 2009) 2040–2045
5. Schlicht, A., Stuckenschmidt, H.: Peer-to-peer Reasoning for Interlinked Ontologies. In: Proc. of Int. Conference on Web Reasoning and Rule Systems. (2009)
6. Aslani, M., Haarslev, V.: Towards parallel classifcation of tboxes. In: Proc. of the 2010 International Workshop on Description Logics (DL-2010). (2010)
7. Haarslev, V., Möller, R.: Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles. In: Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2000). (2000) 273–284
8. Liebig, T., Müller, F.: Parallelizing Tableaux-Based Description Logic Reasoning. In: Proc. of the Int. Workshop on Scalable Semantic Web Systems (SSWS). Volume 4806 of LNCS., Springer (2007) 1135–1144
9. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimising Terminological Reasoning for Expressive Description Logics. Journal of Automated Reasoning **39**(3) (2007) 277–316

[1] `http://qt.nokia.com/`

Some entities are more equal than others: statistical methods to consolidate Linked Data*

Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann
Digital Enterprise Research Institute, National University of Ireland, Galway
{firstname.lastname}@deri.org

Abstract. We propose a method for consolidating entities in RDF data on the Web. Our approach is based on a statistical analysis of the use of predicates and their associated values to identify "quasi"-key properties. Compared to a purely symbolic based approach, we obtain promising results, retrieving more identical entities with a high precision. We also argue that our technique scales well—possibly to the size of the current Web of Data—as opposed to more expensive existing approaches.

1 Introduction

In a distributed and collaborative environment like the current World Wide Web, there can be a lot of redundancy across data sources. While redundancy increases noisy or unnecessary information, it can also be an advantage, in the sense that two descriptions of the same thing can mutually complete and complement each other. However, identifying the *same thing* is not a straight-forward task at all, since different identifiers are used for equal entities scattered across different datasets on the current Web of Data.

In the Semantic Web, identical entities can be made explicit by asserting a `owl:sameAs` (resp. `owl:equivalentClass`, `owl:equivalentProperty`) relations between instances (classes, properties, resp.). Entity consolidation on the Semantic Web – as we view it here – thus boils down to the task of identifying `owl:sameAs` relations between instances which are not explicitly related. In the literature, this task is also sometimes referred to as "record linkage" [1], "duplicate identification" [2], "object consolidation" [3], "instance matching" [4], "link discovery" [5, 6], or "Co-reference resolution" [7].

In this paper, we define techniques for entity consolidation that take advantage of statistical information about the way predicates are used throughout the Web of Data in order to assess whether these predicates, along with the values associated with them, are good candidate for identifying instances or, conversely, discriminating them. For the moment, our technique relies on data described in terms of overlapping vocabularies, i.e. sharing common identical properties; as we focus on scale, we have decided for a deliberately simple approach.

The idea that we try to automatically simulate with our approach is based on the following intuitions:

1. we can conclude that two instances are representing the same real-world entity if they share several common property-value pairs;

* This work is partly funded by Science Foundation Ireland (SFI) project Lion-2 (SFI/08/CE/I1380) and an IRCSET postgraduate scholarship.

2. certain properties are more or less appropriate to disambiguate/consolitate entities;
3. likewise, certain values of some properties are more or less appropriate to disambiguate/consolidate entities.

Particularly, we assume that the necessary information to exploit (Item 2) and (Item 3) can be gathered from statistics about Web data.

As an example, let us consider the case of identifying persons. Two descriptions of unknown persons are representing the same human individual if they describe common properties, such as, eye colour, height, gender, name (Item 1). The gender of a person is usually of limited utility to identify someone, as opposed to the name or address (Item 2). However, if the name is a very common one, such as "Sam Smith" in an English-speaking country, the name property is not enough to identify the person with reasonable certainty (Item 3) and e.g. another property that is non-disscriminating by itself necessarily, such as the gender for instance, may again become discriminating.

Interestingly, these three intuitions can be formalised into an algorithm that we describe in Section 2. Then, we detail its implementation and some improvement to make the process scalable and more efficient in Section 3. In Section 4, we describe a preliminary but promising evaluation of our approach, along with some more general discussion on the feasibility of properly evaluating such a system. Section 5 presents related work and compare it to our approach. In Section 6, we conclude wih important issues still to be solved, possible improvements and future work.

2 Statistical entity consolidation

In this section, we formulate an abstract algorithm for computing a similarity measure on pairs of RDF terms.

Let us first describe some formal notions used throughout this paper. We denote RDF terms by $\mathbf{U}$, $\mathbf{B}$ and $\mathbf{L}$, i.e. the sets of all URIs, blank nodes, and literals, respectively. RDF documents are sets of triples $\langle s\ p\ o\ . \rangle \in \mathbf{B} \cup \mathbf{U} \times \mathbf{U} \times \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$. For a given RDF document G, we denote by sub(G) (resp. pred(G), obj(G)) the set of subjects (resp. predicates, objects) appearing in G. We write RDF documents in the common [1] notation.

Example 1. For illustration purposes, let's consider three documents crawled from the Web containing candidate identifiers for consolidation—*viz.*, `ex1:SamSmith`, `ex2:sam_smith` and `ex3:Sam-Smith`—as follows:

```
ex:SomeDoc dc:creator ex1:SamSmith .
ex1:SamSmith a foaf:Person ; foaf:name "Sam Smith" ;
             foaf:gender "male" ; foaf:homepage ex:JSHompage .
```

[1] `Turtle.http://www.w3.org/TeamSubmission/turtle/`

```
ex:SomeDoc dc:creator ex2:sam_smith .
ex2:sam_smith foaf:name "Dr. Sam J. Smith" ;
              foaf:homepage ex:JSHompage .


ex:SomeOtherDoc dc:creator ex3:Sam-Smith .
ex3:Sam-Smith a foaf:Person ; foaf:name "Sam Smith" ;
              foaf:gender "female" .
```

A human able to interpret the above notation will quickly discern that `ex1:SamSmith` and `ex2:sam_smith` *likely* refer to the same person, and that `ex3:Sam-Smith` is a separate person. In this case, a human will intuitively understand that a single document is unlikely to have two authors with the same (first and last) name, and that two people will rarely share a homepage. Sharing the same name and the same homepage are both good indicators that the first and second entities are referring to the same person. However, class membership such as `foaf:Person` does not particularly indicates uniqueness. A human will also understand that the third Sam Smith is female, and that a person usually only has one unique value for gender—thus, the third entity is distinct from the earlier two.

In the following, we try to formalise the above described intuitions such that we can implement an algorithm for performing entity consolidation similar to our fictitious human consumer from this example. A human naturally has the required experience of the world to draw the above conclusions, whereas a machine does not; thus, we must first derive a means of identifying properties and property value pairs which somehow discriminate an entity: *e.g.*, that different entities rarely have the same value for the `foaf:homepage` property. We must then provide a means of translating our knowledge of properties and values into probabilistic equivalence assertions for entities with shared property-value pairs: *e.g.*, that if two entities share a homepage, then there is a probability of p that they are the same. We must further provide a means of aggregating all p values for the same entity pairs to derive an overall score for an equivalence relation between those two entities. Finally, following similar trains of thought we should be able perform disambiguation of entities—again using our statistical knowledge of the usage of properties—to derive a score indicating the likelihood that two entities are *not* equivalent: *e.g.*, that if two candidates initially deemed likely to be equivalent have a different value for `foaf:gender`, then they are likely not equivalent after all. However, in the present paper we only focus on the consolidation part, leaving disambiguation as future work.

2.1 Web Crawl Dataset

To illustrate the type of results produced in our approach, we use a 20M triple RDF Web crawl for which we offer statistics and later derive some evaluation. This dataset was crawled in late January 2010. We also derive some real examples from the dataset in this section. We refer to this dataset as G_{20M}.

2.2 Property-centric statistics

In this paper, we tackle consolidation by relying purely on the statistical characteristics of properties as observed for a given RDF graph. So, to begin, we formalise some statistical characteristics of properties and property-value pairs which approximately quantify how discriminating these are, i.e., to what degree they "identify" the entity to which they are attached.

Thus, when in what follows we speak of *cardinality* for example, it is important to note that we rather intend the notion of an "observed" cardinality—observed with respect to a given graph—in contrast to, *e.g.*, the cardinality explicitly declared within OWL constructs `owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`, `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`, etc. With this in mind, we now give some preliminary definitions.

Definition 1 (Cardinality). *Let G be an RDF document, p be a property used as a predicate in G and s be a subject in G. The* observed cardinality *(or simply* cardinality*)* of p wrt s in G*, denoted* $\text{Card}_G(p, s)$*, is the cardinality of the set* $\{o \in \text{obj}(G) \mid \langle s\ p\ o\ . \rangle \in G\}$.

Example 2. Take the graph G_{EX} of all triples from Example 1; the cardinality of the property `dc:creator` with respect to the subject `ex:SomeDoc` is given as $\text{Card}_{G_{EX}}(\texttt{dc:creator}, \texttt{ex:SomeDoc}) = 2$.

We see the cardinality as an initial indicator of how suitable a given pair $< p, s >$ is for discriminating an entity identified by the object. Given a set of cardinalities for a given property, we can define the straightforward notion of *average cardinality* for p as the average of all cardinalities observed for p; *viz*:

Definition 2 (Average cardinality). *Let G be an RDF document, and p be a property used as a predicate in G. The* average cardinality of p*, written* $\text{AC}_G(p)$*, is the average of the non-zero cardinalities of p wrt a variable s. Formally,* $\text{AC}_G(p) = \frac{\sum_{s \in \text{sub}(G)} \text{Card}_G(s,p)}{|\{s \in \text{sub}(G) | \langle s\ p\ o\ . \rangle \in G\}|}$.

Example 3. Again given G_{EX}, the average cardinality of the property `dc:creator` is given as $\text{AC}_{G_{EX}}(\texttt{dc:creator}) = 1.5$.

Given a property appearing as a predicate in the graph, the corresponding average cardinality is necessarily a positive value greater than one. We may view the average cardinality as roughly corresponding to the probability that two entities identified by a given object are equivalent if they share a given predicate-subject pair—more succinctly, we could interpret properties with average cardinalities close to one as *quasi-functional*.

Given that RDF graphs preserve direction, we can likewise introduce the dual notion of *inverse cardinality* and *average inverse cardinality*, which intuitively coincide with the above definitions replacing subject with object; *viz.*, :

Definition 3 (Inverse cardinality). *Let G be an RDF document, p a predicate in G and o an object in G. The* inverse cardinality of p wrt o in G *is the cardinality of the set* $\{s \in \text{sub}(G) \mid \langle s\ p\ o\ . \rangle \in G\}$*. This is written* $\text{ICard}_G(p, o)$.

Example 4. Again given G_{EX}, the inverse-cardinality of property `foaf:name` with respect to object `"Sam Smith"` is: $\text{ICard}_{G_{EX}}(\texttt{foaf:name}, \texttt{"Sam Smith"}) = 2$.

Definition 4 (Average inverse cardinality). *Let G be an RDF document, p a predicate in G. The* average inverse cardinality of p *is the average of the non-zero inverse cardinalities of p wrt a variable o. This is written* $\text{AIC}_G(p)$. *Formally,* $\text{AIC}_G(p) = \frac{\sum_{o \in \text{sub}(G)} \text{Card}_G(p,o)}{|\{o \in \text{sub}(G) | \langle s\ p\ o\ . \rangle \in G\}|}$.

Example 5. Again given G_{EX}, the average inverse cardinality of the property `foaf:name` is: $\text{AIC}_{G_{EX}}(\texttt{foaf:name}) = 1.5$.

In analogy to the above said, we may view the inverse cardinality as an initial indicator of how suitable a given $< p, o >$ pair is for discriminating an entity identified by the subject, and see low average inverse cardinality scores as an indicator for *quasi-inverse-functional* properties.

Please note that hereafter, whenever there is no ambiguity, we conveniently omit the name of the graph in index, writing, *e.g.*, $\text{Card}(p, s)$ instead of $\text{Card}_G(p, s)$.

The above indicators are indeed naïve in terms of quantifying the inverse-functional/functional nature of a given property, and require further tailoring.

Strangely, the *absolute* accuracy of the above metrics are contingent on the consistency of naming for entities—the lack of which is the precise motivation for the metrics; *e.g.*, if we see that seven distinct subjects—which in actuality refer to the same book—have a given object-value for the property `ex:isbn`, we would unduely punish `ex:isbn` by deriving a higher score for the average cardinality. However, we would hope that the more important *relative* accuracy of our metrics in a large enough dataset are not so affected—as long as the metrics for our properties are *proportionately* affected by inconsistent naming, we are not so concerned.

In order to remove obvious noise, we must firstly consider the prevalence of blank-nodes in Linked Data and their effect on our metrics: obviously, by their very nature blank-nodes cannot have any naming consistency across Web documents. For example, the social blogging platform hosted on the `livejournal.com` domain exports large volumes of FOAF[2] data describing users, and only infrequently uses URIs to identify entities; users are given unique blank-node identifiers in each document they appear in. Now, *e.g.*, when the same `foaf:weblog` object-value is given for the same user in several different documents, the average inverse cardinality of `foaf:weblog` is severely and disproportionately increased. In order to improve our initial naïve metrics, we can begin them by simply ignoring blank-node objects when computing average cardinalities and, conversely, ignoring blank-node subjects when computing inverse average cardinalities. We denote these adapted metics excluding blank nodes by Card-XB and AIC-XB, respectively.

Along these lines, in Table 1 we present the average inverse cardinality for the top five of those properties in our Web crawl which are explicitly declared to be inverse-functional (i.e. of type `owl:InverseFunctionalProperty`). Following the

[2] `http://foaf-project.org`

above discussion, we would reasonably expect values close to one; we also show the corresponding values when blank-nodes are ignored as above. Somewhat confirming our suspicion, we can observe that, *e.g.*, the AIC for `foaf:weblog` becomes more accurate when blank-nodes are ignored. We also note that `foaf:mbox` still has a high AIC-XB value due to one source which exports the same `foaf:mbox` values for numerous diverse URI subjects.[3]

IFP	Occurrences	AIC	AIC-XB
`foaf:weblog`	113,091	1.978	1.007
`foaf:mbox_sha1sum`	74,525	1.039	1.014
`foaf:homepage`	72,941	1.016	1.004
`contact:mailbox`	1,272	6.144	1
`foaf:mbox`	1,113	2.338	2.006

Table 1. Average inverse-cardinalities for the top five instantiated properties asserted to be inverse-functional.

We provide similar analysis in Table 2, giving average cardinalities for declared `owl:FunctionalProperti`es. Again we note that the values approximate one, but we observe that the results are generally less affected by blank-nodes.

FP	Occurrences	AC	AC-XB
`foaf:primaryTopic`	69,072	1.066	1.065
`loc:address`	2,540	1	1
`loc:name`	2,540	1	1
`loc:phone`	2,540	1	1
`foaf:gender`	1,513	1.001	1.001

Table 2. Average cardinalities for the top five instantiated properties asserted to be functional.

Another problem which requires consideration in our metrics is that of incomplete knowledge: given the fact that less observations derive a lower AC/AIC score for a property, we should be more conservative in using less observed properties for consolidation. Thus, we introduce the notion of an *adjusted average cardinality*, where we use a standard credibility formula to dampen averages derived from relatively few observations towards a more conservative mean value [8].

Definition 5 (Adjusted Average Cardinality). *Let p be a property appearing as a predicate in the graph. The* adjusted average cardinality of p *is then* $\text{AAC}(p) = \frac{\text{AC}(p) \times n_{\overleftarrow{p}} + \overline{\text{AC}} \times \overleftarrow{n}}{n_{\overleftarrow{p}} + \overleftarrow{n}}$ *where $n_{\overleftarrow{p}}$ is the number of distinct subjects that appear in a triple with p as a predicate, $\overline{\text{AC}}$ is the average cardinality for all predicate-subject pairs, and $\overleftarrow{n}$ is the average number of distinct subjects for all predicates in the graph.*

Note that above, it may be more intuitive to think of $n_{\overleftarrow{p}}$ as corresponding to the number of observed cardinalities used to derive $\text{AC}(p)$. The above credibility

[3] `http://rdfweb.org/2003/02/28/cwm-crawler-output.rdf`

formula ensures that for AC values derived from a low number of observations ($n_{\overline{p}} \ll \overleftarrow{n}$), the adjusted AC value is more influenced by the mean $\overline{\text{AC}}$ value than the observed value AC(p); conversely, when $n_{\overline{p}} \gg \overleftarrow{n}$, the observed AC($p$) value has more influence. From our dataset, for the AAC we observed a value for $\overleftarrow{n}$ of 3985, and a value for $\overline{\text{AC}}$ of 1.153.

We define Adjusted AIC analogously, where $\overline{\text{AIC}}$ denotes the average cardinality for all predicate-object pairs and $\overrightarrow{n}$ is the average number of distinct objects for all predicates in the graph. From our dataset, for the AAIC we observed a value for $\overrightarrow{n}$ of 754, and a value for $\overline{\text{AIC}}$ of 6.094.

Example 6. From G_{20M}, for property `rel:childOf`, AIC(`rel:childOf`)=1.414 and $n_{\overrightarrow{\texttt{rel:childOf}}}$=74. Then, AAIC(`rel:childOf`)=$\frac{74\times1.414+6.094\times754}{74+754}$=5.85: a conservative score reflecting the lack of observations for `rel:childof`.

Taking property `foaf:name`, AIC(`foaf:name`)=1.161 and $n_{\overrightarrow{\texttt{foaf:name}}}$=66,244. Then, AAIC(`foaf:name`)=$\frac{66{,}244\times1.161+6.094\times754}{66{,}244+754}$=1.293: a more confident score reflecting the wealth of observations for `foaf:name`.

2.3 Computing confidence for entity equivalences

We now want to use the cardinalities, inverse cardinalities, AAC and AAIC values of properties and values that are shared by two entities to derive some score indicating the likelihood that those two entities are equivalent; referring back to our running example, the instances `ex1:SamSmith` and `ex2:sam_smith` share the object-value `ex:JSHomepage` for property `foaf:homepage` and the subject-value `ex:SomeDoc` for the property `dc:creator`—similarly, `ex1:SamSmith` and `ex3:Sam-Smith` share the object-value `"Sam Smith"` for property `foaf:name`. To do this, we need a metric which combines the (inverse) cardinality and the AA(I)C score for a given property-value pair, where the former value indicates the "uniqueness" of the value for the property, and the latter value gives a more general indication of the (inverse-) functional nature of the property.

We start by assigning a coefficient to each pair $\langle p, o\rangle$ and each pair $\langle p, s\rangle$ that occur in the dataset, where the coefficient is an indicator of how much the pair helps determining the identity of an entity. In particular, for the purposes of later aggregation, we require the coefficient to be a positive value less than one. We determine the coefficient for a $\langle p, s\rangle$ pair as $C(p, s) = \frac{1}{\text{Card}(p,s)\times\text{AAC}(p)}$, and the the coefficient for $\langle p, o\rangle$ as $C^-(p, o) = \frac{1}{\text{ICard}(p,o)\times\text{AAIC}(p)}$.

Example 7. Take $G_{EX'}$ as a version of G_{20M} which contains G_{EX}—essentially, we want to refer to the running example using real statistics from our evaluation. Take AAIC$_{G_{EX'}}$(`foaf:name`)=1.293 as before.

Now, let us speculate that ICard$_{G_{EX'}}$(`foaf:name`, `"Sam Smith"`) = 7, reflecting in this example that "Sam Smith" is somehow a relatively common name. Then, C^-(`foaf:name`, `"Sam Smith"`) = $\frac{1}{7\times1.293} = 0.11$.

Now, speculate that ICard$_{G_{EX'}}$(`foaf:name`, `"Dr. Sam J. Smith"`) = 2, reflecting in this example that the name "Dr. Sam J. Smith" is more rare. Then, C^-(`foaf:name`, `"Dr. Sam J. Smith"`) = $\frac{1}{2\times1.293} = 0.387$.

With coefficients for each property-value pair at hand, we can now derive an aggregated confidence score for entity equivalences. To this end, we define the following aggregation function:

Definition 6 (Aggregated Confidence Score). *Let $Z = (z_1, \ldots z_n)$ be a non-empty n-tuple such that $Z \in [0,1]^n$ and let* $\max \in [0,1]$. *The aggregated confidence value* $\text{ACS}(Z, \max)$ *is computed iteratively: starting with* $\text{ACS}^0 = 0$, *then for each* $k = 1 \ldots n$, $\text{ACS}^k = (\max - \text{ACS}^{k-1})z_k + \text{ACS}^{k-1}$.

The above confidence function is commutative (wrt. the order of z_i, z_j) and produces a value between 0 and max inclusive. Taking max as 1, the main idea is to view Z as a list of probabilistic scores for a given observation, and that each successive score ACS^k reduces the uncertainty $1 - \text{ACS}^{k-1}$ by a product of the current observation z_k—we parameterise max for full flexibility of the aggregation function. Also, the function gives higher weight to more certain observations. Indeed, take $Z_a = (0.5, 0.5)$ and $Z_b = (0.9, 0.1)$; $\text{ACS}(Z_a, 1) = (1 - 0.5) \times 0.5 + 0.5 = 0.75$ whereas $\text{ACS}(Z_b, 1) = (1 - 0.1) \times 0.9 + 0.1 = 0.91$.

To compute the aggregated confidence score for the equivalence of two entities e_1, e_2, we first define the sequence of subject equivalence coefficients $s^{e_1,e_2} = (s_1^{e_1,e_2}, \ldots, s_n^{e_1,e_2})$ as an ordering of the multiset $\{C^-(p,o) \mid \langle e_1\ p\ o\ .\rangle \in G \wedge \langle e_2\ p\ o\ .\rangle \in G\}$—that is, the coefficients for pairs $\langle p, o\rangle$ that appear in a triple with subject e_1 as well as in a triple with subject e_2. We define the sequence of object equivalence coefficients $o^{e_1,e_2} = (o_1^{e_1,e_2}, \ldots, o_n^{e_1,e_2})$ analogously via $C(p,s)$.

Let Z_{e_1,e_2} be the concatenation of the sequences s^{e_1,e_2} and o^{e_1,e_2}, that is, $Z_{e1,e2}$ represents the confidences derived from the coefficients of all property-value pairs shared by the two entities. We could now naïvely compute the aggregated confidence score as $\text{ACS}(Z_{e1,e2}, 1)$.

Example 8. Again take $G_{EX'}$, where $\text{AAIC}_{G_{EX'}}(\texttt{foaf:homepage}) = 1.068$ and $\text{AAC}_{G_{EX'}}(\texttt{dc:creator}) = 1.214$. Further, let us assume $\text{ICard}_{G_{EX'}}(\texttt{foaf:homepage}, \texttt{ex:JSHompage}) = 2$ and $\text{Card}_{G_{EX'}}(\texttt{dc:creator}, \texttt{ex:SomeDoc}) = 2$. As before, we can determine $C^-(\texttt{foaf:homepage}, \texttt{ex:JSHompage}) = 0.468$ and $C(\texttt{dc:creator}, \texttt{ex:SomeDoc}) = 0.412$.

Now, taking `ex1:SamSmith` and `ex2:sam_smith` as candidates for consolidation, we can determine $Z_{\texttt{ex1:SamSmith},\texttt{ex2:sam\_smith}} = (0.468, 0.412)$, and finally compute $\text{ACS}(Z_{\texttt{ex1:SamSmith},\texttt{ex2:sam\_smith}}, 1)$=0.687.

However, the above aggregation is still too naïve for Web data in that it assumes that observations based on property-value pairs are completely independent. As a counter-example, we present Figure 1 which shows a real sample taken from our crawl in which we see two people share some relation to six distinct subject/object values. We observe a clear correlation between these properties.

Firstly, we must consider that two entities which share at least one value for a given property are more likely to share subsequent values; thus, we cannot view the subsequent readings as independent observations, and must take into account possible correlation: *e.g.*, two people who have co-authored at least one paper together are more likely to co-author more. Thus, as a counter measure,

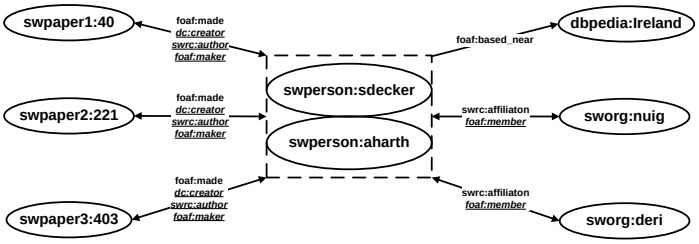


Fig. 1. Real example of inter- and intra-property correlation

for the observed shared property-value pairs for e_1 and e_2, we first aggregate the values for each property p_k (in each direction) separately using the above aggregation function: during this aggregation, we set the max value to $\frac{1}{AC(p_k)}$ or $\frac{1}{AIC(p_k)}$ respectively. Thus, for the example presented in Figure 1, we would only allow, *e.g.*, `dc:creator` to contribute a total value of 0.824. We then perform the aggregation function again over the individual scores of all properties (in each direction).

Aside from correlation for values on a single property, there may also be correlation between different properties—*e.g.*, sub-properties or inverse-properties—which relate two entities to the same external literal or entity. Thus, we prune our observations whereby if we have multiple properties connected to the same term, we keep the property with the lowest AC(p) or AIC(p) value for either direction, and remove consideration of all other properties.

The above two steps to counter-act "obvious" correlation reduced the aggregated confidence scores for the two entities presented in Figure 1 from 0.969 in the naïve case, to 0.781. Admittedly, the new confidence is still quite high—one could further try to detect and account for less obvious forms of correlation such as between a person's affiliation, location and co-authors. However, such considerations are outside of the current more preliminary scope.

3 Implementation

In order to simplify discussion of our implementation, we solely refer to the calculations based on AIC until necessary. Calculations based on AC are directly analogous, where object and subject are simply swapped.

We wish to see our methods used at scale over Linked Data, thus we attempt to use scalable operations to implement our statistical analysis: specifically, we rely mainly on sorts and scans. Data is stored in N-Triples (or possibly N-Quads) format in a flat GZipped compressed file.

Assuming an input unsorted dataset, our first step is to sort the data according to the following lexicographic order (using a merge-sort):

$$(p, o, s)$$

The data is thus grouped according to common p values, and further according to common p, o pairs. Thus, we can calculate the inverse-cardinality for each p, o by means of a scan. Further, by storing the distribution of inverse-cardinalities observed for a given property, we can similarly compute the average inverse cardinality for each property on the fly. Thus, we perform a single scan of the ordered data and extract all of the cardinality information needed for the proceeding steps, as well as the $\overleftarrow{n}$ and $\overline{\text{AC}}$ figures required for the credibility formula.

We can then perform a second scan of the same data, this time using the statistics produced in the first scan to derive initial confidence scores for each individual *po* pair. That is to say, we can use the $AIC(p)$ and $\text{ICard}(p, o)$ to compute $C^-(p, o)$ values, and propagate these values as initial indicators of equivalence for subjects with the same $\langle p, o\rangle$. Thus, after the second scan we produce the following tuples:

$$(e_1, e_2, C^-(p, o), p, o, -)$$

These tuples are written again to a new compressed file (in general N-Triple form); note that the '$-$' is simply to indicate direction of the observation.

Applying the exact same process over data ordered by: (p, s, o), we can also derive tuples of the form:

$$(e_1, e_2, C(p, s), p, s, +)$$

Note that we do not produce reflexive or symmetric versions of the above tuples: for the above tuples, e_1 will always be less than e_2 with respect to the given lexicographical order which allows us to halve the set of tuples, while ensuring consistency in tuple "naming". Indeed, the production of such tuples is quadratic with respect to the given input which naïvely could seriously hamper scalability we aim for. In order to illustrate this, Figure 2(a) and Figure 2(b) show the cumulative increase in tuples when considering increasing sizes of "equivalence classes" derived for increasingly common p, o and p, s pairs respectively. Conveniently however, the increased equivalence class sizes corresponds to a higher inverse-cardinality/cardinality values, which implies that the common $\langle p, o\rangle/\langle p, s\rangle$ pairs which produce the larger equivalence classes are in any case useless for consolidation in our scenario. For the moment, we implement an arbitrary threshold and throw away equivalence tuples derived from $\langle p, o\rangle/\langle p, s\rangle$ pairs with s/o values greater than 100.

Finally, both incomplete sets of tuples can then be merge-sorted to produce a file grouped by e_1 and e_2. The sorted tuples can then be scanned, with the above aggregation functions being applied for each $\langle e_1, e_2\rangle$ pair.

We deem the above methods to be relatively scalable—with the caveat of quadratic equivalence tuples being produced—again, with a sensible threshold, such explosion of output can be mitigated. In any case, we admittedly have yet to test our methods with respect to performance or scale on larger datasets with varying thresholds. For the moment, we focus on some quality evaluation to ensure that our approach derives some reasonable results.

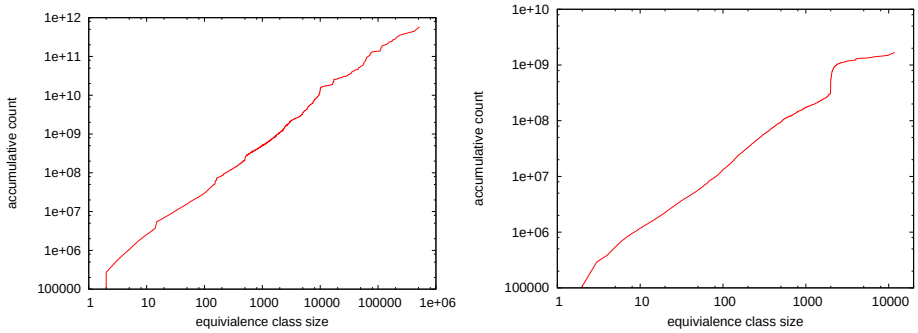


Fig. 2. Cumulative increase in tuples when considering increasing sizes of "equivalence classes" derived for increasingly common p, o and p, s pairs respectively.

4 Quality evaluation

The evaluation of our approach is problematic because there is no existing benchmark for consolidation of Web data. We nonetheless tried two different approaches.

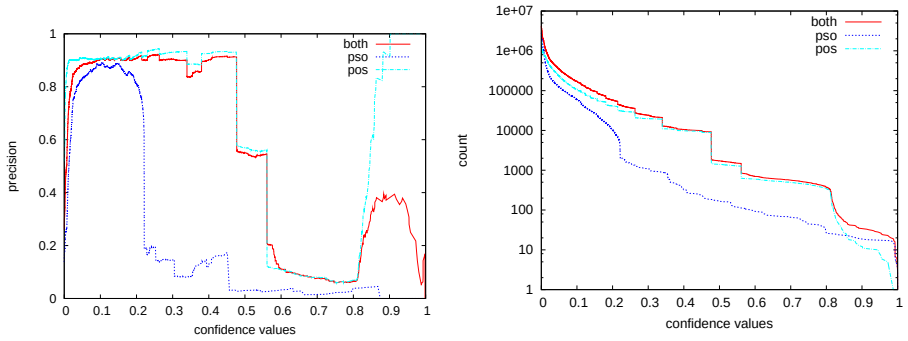
Firstly, we extract our own "best-effort" benchmark from our crawl of 20 million triples by the following process:

- we extract asserted `owl:sameAs` statements and infer additional `owl:sameAs` statements using the same technique as in [3]—a single iteration of reasoning using `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` assertions;
- we separate all `owl:sameAs` statements and additionally compute the transitive closure thereof;
- we prune the dataset by keeping only triples which have either a subject or an object that appears in a `owl:sameAs` statement;
- again, we discard the `owl:sameAs` statements which relate an entity for which we have no information;
- we again finally prune the dataset removing triples for which the subject or object do not have `owl:sameAs` statements.

The resulting set of `owl:sameAs` statements contains 36,134,230 transitively closed, non-symmetric, non-reflexive (reflecting the nature of the same-as output of our statistical approach) `owl:sameAs` statements over 87,586 entities. The evaluation data consists of 5,622,898 triples. We view the derived asserted/inferred `owl:sameAs` statements as a partial ground-truth for our quality evaluation: please note that we are aware of the somewhat ironic nature of our evaluation approach—if we apply our previous work on reasoning, we would achieve a perfect 100% recall and 100% precision. However, again this evaluation is best-effort, and is intended in this preliminary analysis to present illustrative statistics about the precision of our approach in the spirit of a proof-of-concept.

Along these lines, in Figure 3(a) we present the precision of our approach considering AIC values, AC values, and both values combined. Indeed, our precision is quite high at even low levels of confidence, reaching roughly 92% at a confidence value of 0.26. However, our approach suffers from deriving a small number of incorrect equivalences at high confidence. Severe drops in precision are due to the derivation of large numbers of correct inferences at an exact precision; *e.g.*, we derive 630 correct inferences at the precise value of 0.6777389199225334—all uniformly described entities found in the aforementioned `livejournal.com` domain. Thus, once we go above that threshold, the precision severely drops. Essentially, large volume equivalences are derived at lower confidence values, and incorrect equivalences between entities described in smaller exporters are derived at higher confidence values. Figure 3(b) is presented for cross-reference, where the amount of remaining correct inferences drop in correlation with the drops in precision from Figure 3(a). Interestingly, from Figure 3(a) we can conclude that considering AIC values alone approximates consideration of both directions.

With respect to recall, we observed a value of about 3% with respect to the transitively closed ground truth. However, one should note that we do not perform any transitive closure over the output of our statistically derived equivalences, and thus it is difficult to derive an adequate recall comparison.



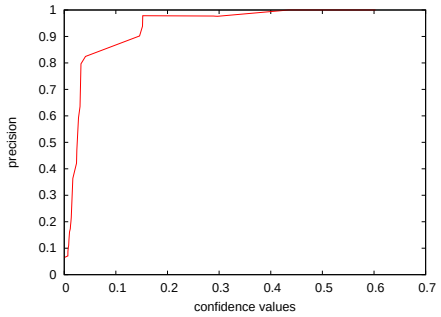
(a) Precision of our approach considering AIC (pos), AC (ps) and both, with respect to different threshold of confidence

(b) Correct equivalences found considering AIC (pos), AC (ps) and both, with respect to different levels of confidence

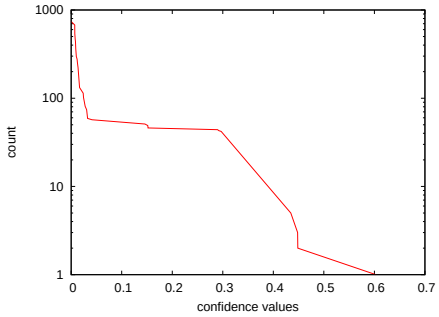
In our second evaluation, we used our consolidation approach as an instance matching tool by selecting only the consolidation that matches named terms from two distinct datasets. This method has the merit of being comparable to other instance matching algorithms over the reference datasets of the Ontology Alignment Evaluation Initiative[4]. The OAEI offers a well established competition in the ontology matching community, and an instance matching track was added in 2009. The drawback of this method is that the datasets used are very homogeneous (3 sets of bibliographic data) and are using the same terms in a very similar

[4] `OAEI.http://oaei.ontologymatching.org/`

way to each other. Therefore, they are not representative of what is really found on the Web of Data. The results in Figure 4 shows that we get a much lower recall than specialised instance matching tools (cf. [9], Fig. 12, p40). We do not consider that this demonstrate a flaw of our approach. On the contrary, we think that it shows the limits of evaluating a generic consolidation approach with a specific, small-scale instance matching dataset. Unfortunately, a solid evaluation standard for entity consolidation is yet to be devised. Our previous home-made benchmark is an attempt in that direction. In this experiment, AC did not contribute at all to the overall confidence, because of the particular morphology of the data.



(c) Precision in function of the threshold of confidence



(d) Correct equivalences found in function of the threshold of confidence

5 Related work

In our previous work, we used reasoning (functional properties, inverse functional properties, cardinality restrictions) to consolidate Web data with limitation both in terms of precision and recall [3]. Entity consolidation has an older related stream of research relating largely to databases, with work under the names of record linkage, instance fusion, and duplicate identification; cf. [1, 10, 11] and a survey at [2]. Due to the lack of formal specification for determining equivalences, these older approaches are mostly concerned with probabilistic methods. Bouquet et al. [12] motivate the problem of (re)using common identifiers as one of the pillars of the Semantic Web, and provide a framework and fuzzy matching algorithms to fuse identifiers. Online systems such as Sig.Ma[5], rkbexplorer[6], and ObjectCoref offer on-demand querying for `owl:sameAs` relations found for a given input URI, which they internally compute and store. Another related field which gained more recent attention is Instance matching. Some references include matching database instances [13], domain-dependent similarity of instances [14, 15], [4], instance matching and linking guided by a "linking language" (Silk) for

[5] `http://sig.ma`
[6] `http://www.rkbexplorer.com/sameAs/`

Linked Data [5]. Also, in 2009, the Ontology Alignment Evaluation Initiative[7] has introduced a new test track on instance matching[8].

6 Discussion and conclusion

Indeed, our work is quite preliminary, and there are many open questions. Firstly, in order for such an approach to be proven useful, we would need to demonstrate that the statistical approach presented can generate additional equivalence relations than standard reasoning approaches. Such was not possible given the nature of our evaluation setups. In theory however, we believe that the presented approach should be able to conclude additional equivalences, and in future work we would need to devise a means of evaluating such. Similarly, we should also incorporate reasoning approaches into the current statistical model, developing a hybrid approach which hopefully generates more equivalences.

Perhaps a more interesting use-case for statistical approaches is for disambiguating entities: that is, stating that two entities are different. Such `owl:differentFrom` relations are rarely specified on the Web—they can however be inferred from, *e.g.*, more common `owl:disjointWith` assertions. Given a set of candidate equivalences derived through reasoning, statistical or hybrid approaches, disambiguation can be applied to improve precision of results; reasoning on, *e.g.*, `owl:InverseFunctionalProperty` assertions is known to be imprecise [3]—clearly, our approach could also benefit from some disambiguation post-processing. Indeed, one could consider an iterative approach, where the confidence scores for equivalence and difference are iteratively refined—and statistics are iteratively made more accurate—until a satisfactory fixpoint.

Further, we would intend to evaluate the performance characteristics of our approach on larger datasets, with the aim of applying the analysis over a dataset in the order of a billion triples. Again, our approach is based on a scalable substrate of sorts and scans, and so we would see this as a feasible goal.

With respect to improving the algorithms presented herein, we would need to consider more advanced topics. Perhaps the most important is the consideration of the source of data when deriving statistics. The statistics for usage of properties is heavily influenced by large RDF exporters on the Web. Most of the incorrect highly-confident equivalences were the result of applying such statistics over smaller heterogeneous sources. One might argue that there currently is not enough heterogeneous Linked Data to give enough confidence for such statistical approaches—the "reasonable ineffectiveness of Linked Data" if you will; however, we should still attempt to consider some notion of a "dataset" as a grouping of uniform RDF data—*e.g.*, published by the same exporter—and consider a weighted version of our statistics which includes such a concept.

Also, we would have to look at deriving some form of transitive closure over the 'fuzzy' equivalences produced to improve recall. The exact nature of such a closure is the topic for future research. Similarly, detection of some notion of correlation

[7] `OAEI.http://oaei.ontologymatching.org/`
[8] Instance data matching. `http://www.scharffe.fr/events/oaei2009/`

between properties—besides the more obvious cases already discussed—is worthy of further investigation, and would be useful to ensure more sensible aggregation of confidence scores. Other topics, such as fuzzy string matching techniques or string-normalisation pre-processing, would also be worth further analysis.

To summarise, we defined a new approach towards consolidating data in a very heterogeneous environment (the Semantic Web at large). We have barely scratched the surface but can already attest that the results are promising.

References

1. Newcombe, H.B., Kennedy, J.M., Axford, S.J., James, A.P.: Automatic Linkage of Vital Records: Computers can be used to extract "follow-up" statistics of families from files of routine records. Science **130**(3381) (1959) 954–959
2. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Trans. on Knowl. & Data Eng. **19**(1) (2007) 1–16
3. Hogan, A., Harth, A., Decker, S.: Performing Object Consolidation on the Semantic Web Data Graph. In: Proc. of the WWW2007 Workshop I^3. Volume 249 of CEUR Workshop Proceedings., CEUR (2007)
4. Castano, S., Ferrara, A., Montanelli, S., Lorusso, D.: Instance Matching for Ontology Population. In: Proc. of SEBD 2008. (2008) 121–132
5. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: Proc. of ISWC 2009. Volume 5823 of LNCS., Springer (2009) 650–665
6. Hassanzadeh, O., Lim, L., Kementsietsidis, A., Wang, M.: A Declarative Framework for Semantic Link Discovery over Relational Data. In: Proc. of WWW 2009, ACM Press (2009) 1101–1102
7. Glaser, H., Jaffri, A., Millard, I.: Managing Co-reference on the Semantic Web. In: Proc. of LDOW 2009. (2009)
8. Whitney, A.W.: The theory of experience rating. In: Proceedings of the Casualty Actuarial Society. Volume 4. (1918) 274–292
9. Euzenat, J., Ferrara, A., Hollink, L., Isaac, A., Joslyn, C., Malaisé, V., Meilicke, C., Nikolov, A., Pane, J., Sabou, M., Scharffe, F., Shvaiko, P., Spiliopoulos, V., Stuckenschmidt, H., Sváb-Zamazal, O., Svátek, V., dos Santos, C.T., Vouros, G.A., Wang, S.: Results of the Ontology Alignment Evaluation Initiative 2009. In: Proc. of OM 2009. Volume 551 of CEUR., CEUR (2009)
10. Michalowski, M., Thakkar, S., Knoblock, C.A.: Exploiting Secondary Sources for Automatic Object Consolidation. In: Proceeding of 2003 KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation. (2003) 34–36
11. Chen, Z., Kalashnikov, D.V., Mehrotra, S.: Exploiting relationships for object consolidation. In: Proc. of IQIS 2005, ACM Press (2005) 47–58
12. Bouquet, P., Stoermer, H., Mancioppi, M., Giacomuzzi, D.: OkkaM: Towards a Solution to the "Identity Crisis" on the Semantic Web. In: Proc. of SWAP 2006. Volume 201 of CEUR Workshop Proceedings., CEUR (2006)
13. Bernstein, P.A., Melnik, S., Mork, P.: Interactive Schema Translation with Instance-Level Mappings. In: Proc. of VLDB 2005, ACM Press (2005) 1283–1286
14. Albertoni, R., Martino, M.D.: Semantic Similarity of Ontology Instances Tailored on the Application Context. In: Proc. of OTM 2006, Part I. Volume 4275 of LNCS., Springer (2006) 1020–1038
15. Albertoni, R., Martino, M.D.: Asymmetric and Context-Dependent Semantic Similarity among Ontology Instances. Jour. on Data Semantics **4900**(10) (2008) 1–30