OBDA vs Forward Chaining: the ForBackBench Framework

Afnan Alhazmi[1], George Konstantinidis[1]

[1]University of Southampton, University Road, Southampton, SO17 1BJ, United Kingdom

Abstract

Query answering in Data Integration/Exchange has been extensively studied using two different approaches of reasoning: forward-chaining (or chase) algorithms and backward-chaining algorithms (or query rewriting). These two approaches have been explored separately in terms of algorithms, practical implementations, and benchmarks. While there are numerous available comparisons within the same approach, there is no framework to compare solutions across both approaches. In this paper, we demonstrate ForBackBench, a framework that provides the web interface and infrastructure for cross-approach comparisons, and automated graph plotting, containing a variety of tools that help explore the differences between chasing and query-rewriting.

Keywords

Query answering, Query rewriting, Chase, Data integration, OBDA, Dependencies, Benchmark.

1. Introduction

In Data Integration (DI) and Data Exchange (DE) heterogeneous data sources are mapped to a mediated schema or an ontology via schema-mappings. There are different formats for these mappings but both these and further database constraints, or ontology axioms, can be captured using implications known as tuple-generating dependencies (TGDs). The most common problem in these settings is how to answer a query posed over the mediating schema by using data in the data sources while taking the constraints into account [1, 2].

Data Exchange (DE) focuses on populating a centralized warehouse, to be made available for querying, with source data, via running the *chase* algorithm. This setting usually considers: a set of source schemas and a target centralized schema, *source-to-target* TGDs that express mappings between the source and the target, *target* TGDs that express dependencies on the target schema, and also a query on the target schema [2, 3].

Data Integration (DI), or virtual integration, on the other hand is usually associated with query answering via query rewriting. In DI and related areas such as Ontology Mediated Query Answering (OMQA) or Ontology Based Data Access (OBDA), backward-chaining algorithms start from the query and process the ontology axioms backwards to generate a query rewriting that incorporates the ontology entailments. Depending on the complexity of the mapping language, the rewriting can be unfolded and executed over the source schema directly. OBDA usually considers the DL-Lite$_R$ family of description logics and the associated OWL2 QL pro-

ISWC2022: The 21st International Semantic Web Conference, October 23–27, 2022, Hangzhou, China

a.alhazmi@soton.ac.uk (A. Alhazmi); g.konstantinidis@soton.ac.uk (G. Konstantinidis)

file [4]. In Description Logic terms, this setting involves a DL-Lite T-Box representing the target ontology, an (optional) A-Box containing data, a query and mappings between the data sources and the target ontology [5].

Both areas have seen a rise in algorithms, practical implementations, and benchmarks. We have developed ForBackBench, a single framework to provide comparison and analysis for systems across both forward- and backward-chaining (for the scenarios that both approaches support/terminate). Providing a unified framework allows us to investigate systems' limits and their interplay. We also make it easier to answer a query by combining algorithms from both approaches, for example, using the chase on the source-to-target mappings (not the data) to produce an equivalent setting with chased mappings and without target dependencies, then using a query rewriting algorithm to answer the query over the new source-to-target mappings (see [6]). We are complementary to other recent frameworks, such as the GTFS-Madrid-Bench framework [7] which addresses the problem of heterogeneity and different formats of the data sources, benchmarking only backward-chaining systems and not considering the target reasoning part of the problem. In this paper, we present our framework infrastructure, set of tools, and functionalities via a user web interface. For the full details see [6].

2. Framework Overview

ForBackBench comes with a set of common query-answering systems, as well as common experimental scenarios, and also a set of integrated converter, generation and management tools to support the evaluation and comparison process, allowing for automated testing, plotting and easy future extension. It supports a variety of languages: TGDs and queries in ChaseBench [3] or Rulewerk [8] formats, OWL, RDF, OBDA-mappings [9], SPARQL, and SQL. The ForBackBench framework is made available as an open-source project[1]. Figure 1 presents a high-level overview of our framework architecture. It consists of six main components, subsequently explained.

Scenarios Conversion Engine. This component of the framework is responsible for producing a *cross-approach* scenario from uploaded forward- or backward-chaining scenarios or pre-existing ones taken form literature (see [6]). It can parse and recognise ChaseBench or OBDA scenarios. ChaseBench scenarios usually come with TGD dependencies and queries in ChaseBench or SQL format, while OBDA scenarios come with OWL ontologies, SPARQL queries and potentially OBDA mappings. The engine then translates the given scenario to scenarios containing all necessary files for all types of systems; this universal set of files and inputs is called a cross-approach scenario. A theoretical basis for our translation between DL-Lite$_R$ and TGDs is developed in [6].

Data/Mapping Generator. If mappings are not being given as part of the scenario, we randomly generate source-to-target TGDs and then use the source-to-target dependencies to translate to OBDA-mappings. Our generated source-to-target dependencies will be either very simple (one-to-one) mappings, or they will be mapping a join query over the sources to a target ontology concept or role (GAV), or a source table to a join query over the target ontology/schema (LAV). As well, if not given, we generate data. The data generator produces CSV data files for the source schema with four different dataset sizes (small, medium, large, and very large ranging

[1] https://github.com/georgeKon/ForBackBench

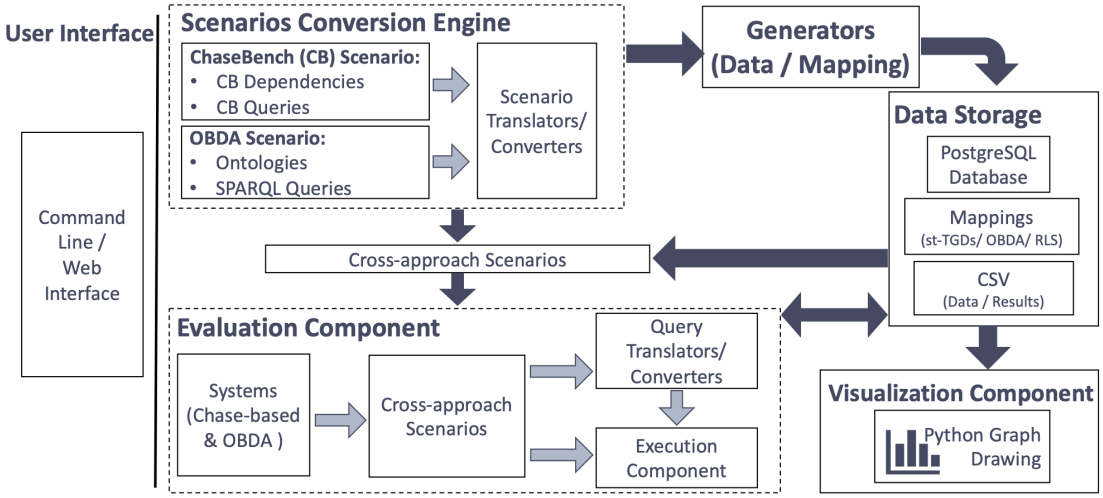

Figure 1: ForBackBench Framework Architecture

from 500 to 1, 000, 000 tuples per table). The output of our conversion engine and our generator is a cross-approach scenario that contains dependencies (in TGD/ChaseBench, RLS, or OWL formats), queries (SPARQL, SQL, ChaseBench), source/target schema (SQL, ChaseBench), OBDA Mappings, and data in CSV files, which is stored in Data Storage. Our choice to support the OBDA-mapping language is driven by our intention to first include the Ontop system [9] in our framework, which is universally considered as one of the most mature systems. We are currently working on extending our framework to support other mapping languages such as R2RML. Our translators and converters can also be used separately as standalone jar files from our repository[2] which contains full description of their functionality and usage.

Evaluation Component. The evaluation component is responsible for automating the process of running end-to-end evaluation and comparisons for query-answering systems on the cross-approach scenarios generated by the conversion engine. The execution component in Fig. 1 is the collection of scripts dedicated to this task. The process starts by selecting a scenario and some of the integrated query-answering systems. First, it runs the "Experiment Run" script, which will invoke the "Database Configuration" script to build a temporary database and then load the data from CSV files into this database. Second, for each query in the scenario, it invokes the "Query Run" script, which runs experiments repeatedly up to the number of times selected. Each run will compute intermediate times as well as the entire time each system takes and store the those performance measurements as well as query results in CSV files. The framework integrates numerous systems from both reasoning approaches (RDFox [10], Rulewerk [8], Llunatic [11], Rapid [12], IQAROS [13], Graal [14], GQR [15], OntopR, Ontop[9], and ChaseGQR [16]).

Data Storage. This component contains the data for each scenario and the results of each experiment as CSV files as well as the PostgreSQL database that is created for the scenario during each experiment. In addition, it stores all files for the cross-approach scenarios.

[2]https://github.com/georgeKon/ForBackBench/tree/main/utilityTools

Visualization Component. We offer python scripts that take experiment results as CSV files and generate a graph with systems' performance. Each system's performance is depicted with a bar whose different segments are coloured differently. The entire bar shows end-to-end time but its subparts show fine-grained times for loading/pre-processing, chasing, rewriting, converting or executing.
Graphical User Interface. ForBackBench provides a user-friendly web interface with two main functionalities. The first is uploading a ChaseBench or OBDA scenario. This will trigger the scenario conversion engine to produce a cross-approach scenario with appropriate data and mapping choices. The user simply has to upload the dependency files (TGDs or OWL), schema files in ChaseBench format, and queries (ChaseBench or SPARQL). The web interface offers a full guide about all required files and their naming scheme. The second function is running experiments by selecting the parameters: the scenario, the data size, the schema mapping format, the number of times for running the experiment, and the systems to be compared. Then, after the experiment is finished, the user can explore the results as a graph or download a full results' folder as a zip file that includes the CSV files with all performance measurements and the plotted graph. A demo video is available for the web interface on: https://youtu.be/gZ0iRvzuSZI.

3. Implementation Details

The ForBackBench framework is implemented using JavaScript, shell script, PHP and Java. JavaScript supports the OBDA converter API, and is also is used to build most of the folder structure for OBDA scenarios. Java supports the OWL API and ChaseBench API, which we use to implement most of our translators of ChaseBench scenarios. For timing scripts, the framework uses the date command to capture the accurate fine-grained timings for each process part for each system. The web interface is implemented using PHP that supports running shell commands in the background and returning results.

4. Demonstration

At the conference, we will demonstrate both ChaseBench and OBDA scenarios. The attendees will be able to interact with the framework by uploading a scenario and generating the data for it. As well, the attendees will be able to run an experiment on one of the pre-integrated scenarios or the uploaded scenario, with some or all query-answering systems. Then, the attendees can explore the results as a graph or CSV files. Running our larger experiments on full scenarios might take hours to finish; thus, for easy interaction with the framework, we will provide simplified versions of the pre-existing ChaseBench and OBDA scenarios for the attendees to interact with and explore immediate results.

References

[1] G. Gottlob, G. Orsi, A. Pieris, Ontological query answering via rewriting, in: J. Eder, M. Bielikova, A. M. Tjoa (Eds.), Advances in Databases and Information Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 1–18.

[2] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: semantics and query answering, Theoretical Computer Science 336 (2005) 89–124.
[3] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, E. Tsamoura, Benchmarking the chase, in: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2017, pp. 37–52.
[4] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, G. Vetere, Dl-lite: Practical reasoning for rich dls, in: 2004 International Workshop on Description Logics, CEUR-WS.org, Whistler, BC, Canada, 2004, p. 92.
[5] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The dl-lite family and relations, J. of Artificial Intelligence Research 36 (2009) 1–69.
[6] A. Alhazmi, T. Blount, G. Konstantinidis, Forbackbench: a benchmark for chasing vs. query-rewriting, Proceedings of the VLDB Endowment 15 (2022) 1519–1532.
[7] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, O. Corcho, Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain, Journal of Web Semantics 65 (2020) 100596. doi:https://doi.org/10.1016/j.websem.2020.100596.
[8] D. Carral, I. Dragoste, L. González, C. Jacobs, M. Krötzsch, J. Urbani, Vlog: A rule engine for knowledge graphs, in: C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, F. Gandon (Eds.), The Semantic Web – ISWC 2019, Springer International Publishing, Cham, 2019, pp. 19–35.
[9] M. Rodríguez-Muro, R. Kontchakov, M. Zakharyaschev, Ontology-based data access: Ontop of databases, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web – ISWC 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 558–573.
[10] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, J. Banerjee, Rdfox: A highly-scalable rdf store, in: M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, S. Staab (Eds.), The Semantic Web - ISWC 2015, Springer International Publishing, Cham, 2015, pp. 3–20.
[11] F. Geerts, G. Mecca, P. Papotti, D. Santoro, The llunatic data-cleaning framework, Proceedings of the VLDB Endowment 6 (2013) 625–636.
[12] A. Chortaras, D. Trivela, G. Stamou, Optimized query rewriting for owl 2 ql, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), Automated Deduction – CADE-23, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 192–206.
[13] T. Venetis, G. Stoilos, G. B. Stamou, Incremental query rewriting for owl 2 ql, in: Proceedings of the 2012 International Workshop on Description Logics, Citeseer, Rome, Italy, 2012, pp. 356–610.
[14] J.-F. Baget, M. Leclère, M.-L. Mugnier, S. Rocher, C. Sipieter, Graal: A toolkit for query answering with existential rules, in: N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, D. Roman (Eds.), Rule Technologies: Foundations, Tools, and Applications, Springer International Publishing, Cham, 2015, pp. 328–344.
[15] G. Konstantinidis, J. L. Ambite, Scalable query rewriting: a graph-based approach, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, Athens, Greece, 2011, pp. 97–108.
[16] G. Konstantinidis, J. L. Ambite, Optimizing the chase: Scalable data integration under constraints, Proceedings of the VLDB Endowment 7 (2014) 1869–1880.