ForBackBench: A Benchmark for Chasing vs. Query-Rewriting

Afnan Alhazmi
University of Southampton
a.alhazmi@soton.ac.uk

Tom Blount
University of Southampton
t.blount@soton.ac.uk

George Konstantinidis
University of Southampton
g.konstantinidis@soton.ac.uk

ABSTRACT

The problems of Data Integration/Exchange (DE) and Ontology Based Data Access (OBDA) have been extensively studied across different communities. The underlying problem is common: using a number of differently structured data-sources mapped to a mediating schema/ontology/knowledge-graph, answer a query posed on the latter. In DE, forward-chaining algorithms, collectively known as the chase, transform source data to a new materialised instance that satisfies the ontology and can be directly queried. In OBDA, backward-chaining algorithms rewrite the query over the source schema, taking the ontology into account, in order to execute the rewriting directly on the sources. These two reasoning approaches have seen an individual rise in algorithms, practical implementations, and benchmarks. However, there has not been a principled methodology to compare solutions across both areas. In this paper we provide an original methodology and a benchmark infrastructure — a set of test scenarios, generator and translator tools, and an experimental infrastructure — to allow the translation and execution of a DE/OBDA scenario across areas and among different chase and query-rewriting systems. In the process, we also present a syntactic restriction of linear Tuple Generating Dependencies that precisely captures DL-Lite$_R$, a correspondence previously uninvestigated. We perform cross-approach experiments under a wide range of assumptions, such as the use of different source-to-target mapping languages, shedding light to the interplay between forward- and backward-chaining. Our preliminary results show that, indeed, chase can compete and might overcome query rewriting even in the face of large data especially for complex mapping languages.

PVLDB Reference Format:
Afnan Alhazmi, Tom Blount, and George Konstantinidis. ForBackBench: A Benchmark for Chasing vs. Query-Rewriting . PVLDB, 15(8): 1519-1532, 2022.
doi:10.14778/3529337.3529338

PVLDB Artifact Availability:
The source code of this research paper has been made publicly available at https://github.com/georgeKon/ForBackBench.

1 INTRODUCTION

Data Integration is the problem of providing unified access to multiple, heterogeneous and distributed sources of data. One of the most important aspects of this problem, and the focus of this paper, is efficient query answering in such settings.

In a data integration/exchange scenario, we have a number of distributed data sources, associated schema mappings, target/ontology dependencies, and a query to answer. There are two prevalent approaches to answering the query. Data Exchange (DE) [25] uses "forward-chaining" algorithms to compute the entailment closure of the schema mappings and the ontology axioms. A prevalent formalisation of these algorithms is the *chase* family of algorithms which consolidates the data to a centralised warehouse ready for query answering. Ontology Based Data Access (OBDA) [4, 17], also known as Ontology Mediated Query Answering, query rewriting, or "backward-chaining", processes the ontology axioms and mappings "backwards" (from consequent to premise) in order to rewrite the query such that it can be answered directly over the sources.

Over the last decade there has been considerable interest in improving query-rewriting solutions (e.g., [20, 37, 50, 51, 56, 57, 64]). The motivation for this has been twofold. On one hand, ontology materialisation (that is, forward-chaining solutions) does not necessarily terminate for common OBDA languages (such as OWL 2 QL/DL-Lite[4, 17]), and thus query rewriting has been treated as the only feasible solution. On the other hand, even when materialisation terminates, this can be a very slow process especially over large datasets; that is, one expects that the chase algorithms will be extremely slow with large data as they are "query-agnostic". In fact, the prevalent assumption is that when a query-driven approach is possible, this solution is always preferable [11, 25].

However, it is very often that examples and datasets from the query-rewriting literature are chase-terminating. In addition, modern query-rewriting systems often make use of substantial query-agnostic preprocessing in order to build up more heavyweight 'smart' indices [19, 55], while the cost of this preprocessing is amortised over multiple runs. This process is reminiscent of the forward-chaining approach, and so there might be scenarios (and we show, there are) where this wouldn't deviate much away from the chase in terms of performance. Moreover, forward- and backward-chaining can be used for a multitude of problems in addition to query answering, where the data is relatively small, e.g., query containment [34], query equivalence, query minimisation, etc. In these latter cases the data to reason with consists of formulas rather than triplestores or databases, and the cost of chasing can be vastly reduced.

We therefore identify the need for a cross-approach benchmarking framework that will enable the comparison, integration and deeper analysis of these two reasoning techniques for the scenarios that can be commonly supported by both, under different assumptions. We present our framework, ForBackBench, that follows in the steps of chaseBench [12]; a principled approach to integrate and benchmark chase algorithms. In fact, we find that such a principled approach has been missing from the OBDA space alone - although there have been implementations and comparisons of algorithms that produce rewritings, or complete end-to-end systems that answer queries via rewritings, there has not been a framework that

Proceedings of the VLDB Endowment, Vol. 15, No. 8 ISSN 2150-8097.
doi:10.14778/3529337.3529338

integrates the query-rewriting implementations into end-to-end query-answering solutions to support comparisons of all OBDA. ForBackBench allows the plug-and-play benchmarking of different implementations of isolated parts of the query answering stages.

In addition, a major part of data integration solutions is the support of schema mapping languages. Schema mappings in data integration commonly come in the form of Global-as-view (GAV), Local-as-view (LAV), and Global-and-local-as-view (GLAV) mappings [26, 41, 62]. For chase algorithms all mappings can be reasoned over in the same manner. However, in the context of OBDA mostly GAV mappings have been explicitly considered [54, 58, 61]. Reasoning with GAV is relatively easy (relies on unfolding) [46, 62], while LAV mappings require a more sophisticated algorithm [35, 53]. In order to support the latter in ODBA, a two-phase approach would be needed; first, produce the OBDA rewritings of the query, and second, use a LAV rewriting algorithm to reformulate the OBDA rewtitings into source queries using the mappings. Yet again, ForBackBench allows the independent reuse of different algorithms for these phases and enables the comparison of end-to-end LAV scenarios in OBDA against forward-chaining approaches.

Our contributions are the following:

- We define a new restriction of linear Tuple Generating Dependencies, called PJ-acyclic TGDs, that precisely captures the essential part of DL-Lite$_R$ (in effect, excluding negative axioms which are usually treated differently).
- We present a novel benchmarking framework, ForBackBench, that includes a variety of benchmarking scenarios, data generators, as well as the tools necessary to convert between forward- and backward-chaining approaches, and the experimental infrastructure to run, and automatically plot comparison evaluations. It supports a number of different languages including TGDs and queries in chaseBench or Rulewerk formats, OWL, RDF, OBDA-mappings, SPARQL, and SQL and is built to be fully extensible.
- We use ForBackBench to benchmark several commonly used query-answering systems against a number of common scenarios from literature and build an understanding of the behavior of both approaches in query answering problems, informing the design of new and more efficient methods.
- Our experiments show surprising results among which are that the chase does in fact terminate in almost all of commonly used benchmark examples and scenarios across the OBDA literature and in many cases running the chase is faster than index building on OBDA systems, or even end-to-end query answering in OBDA, with the additional advantage that no further rewriting is required once the chase terminates. Thus, contrary to a common misconception, the chase should not be dismissed out-of-hand; it is still viable and has the most efficient results in many cases.

2 DATA EXCHANGE AND OBDA

In this section, we introduce the problems of conjunctive query answering in data integration/exchange, and the two main approaches to address it: forward- and backward-chaining.

2.1 Certain Conjunctive Query Answering

A database schema is defined as a set of relation schemas $R = \{R_1, R_2, \ldots, R_k\}$, where each relation schema has a relation name R_i and is associated with an arity $n > 0$, denoting the number of the relation's attributes. An expression $R_i(\vec{x})$, where R_i is a predicate or relation name of some arity n and $\vec{x}$ a tuple of variables or constants, is called an atom of arity n. A ground atom or a fact is an atom that only contains constants. A relation instance, or relation, over a relation schema R_i is set of facts with the same arity. A database (or triplestore) instance is a set of facts over a database schema R; that is, the union of relation instances over the relation schemas in R.

We use rule notation for safe Conjunctive Queries (CQs) [1], e.g.: $q(x) \leftarrow R_1(x, y, \text{“8”}), R_2(x, z)$. The *body* of the query, $body(q)$, is the set of atoms in the antecedent while the *head* is the atom in the consequent. Commas between atoms in the body stand for conjunctions. The variables in the head are free or *distinguished* (e.g., x) and the rest are *existential* variables (e.g., y, z). A union of conjunctive queries (UCQs) is a set of Conjunctive Queries whose head atoms are of the same predicate and the same arity. CQs correspond to SELECT-PROJECT-JOIN queries of SQL and to the Basic Graph Patterns (BGP) of SPARQL. Given a database instance I and a CQ q, by $q(I)$ we denote the set of answer tuples of q on I evaluated in the usual way [1].

In our data integration setting, we examine two families of schemas: a source schema under which data is structured and a target/mediating schema, initially with an empty instance, which is intended as a virtual layer for querying. Mappings between the two kinds of schemas as well as ontology axioms/constraints on top of the target schema can be expressed as database dependencies. There are two prevalent types of dependencies: tuple-generating dependencies (TGDs) and equality-generating-dependencies (EGDs), which are of the form (1) and (2) respectively [25].

$$\forall \vec{x}, \vec{y}(\varphi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z})) \tag{1}$$

$$\forall \vec{x}(\varphi(\vec{x}) \rightarrow (x_1 = x_2)) \tag{2}$$

We sometimes omit quantifiers for brevity and readability. Formulas ϕ, and ψ are conjunctions of atoms, i.e., conjunctive queries. When expressing *source-to-target* schema mappings we use TGDs where ϕ is over the source schema and ψ over the target schema. When denoting ontology axioms or *target constraints* we have ϕ and ψ both on the target schema only.

In the rest of the paper we focus only on TGDs; these capture the major parts of both mapping and ontology languages. Linear source-to-target TGDs (st-LTGDs) correspond to LAV mappings in data integration, while full source-to-target TGDs (that is, st-TGDs with no existential variables) capture GAV. General s-t TGDs are known as GLAV. Target TGDs (t-TGDs) that are *linear* (LTGDs), i.e., where the antecedent is a single atom, already capture the essential part of the DL-Lite family of Description Logics [4, 17], and more specifically DL-Lite$_R$ which underpins the QL profile of OWL 2 [15, 45] and is preferred for querying large datasets.

A *DL-Lite$_R$ knowledge base* [4, 17] (KB) is a pair $O = (\mathcal{T}, \mathcal{A})$, composed of: (1) the T-Box, $\mathcal{T}$, which is a set of class inclusion axioms of the form (i) and role inclusions of the form (ii) below, and (2) the A-Box, $\mathcal{A}$, which describes data in the form (iii) of class and property membership assertions.

$$CL \sqsubseteq CR \quad \text{(i)} \qquad\qquad R_0 \sqsubseteq R_1 \sqcap \ldots \sqcap R_n \quad \text{(ii)}$$

$$C(a) \qquad P(a, b) \qquad\qquad \text{(iii)}$$

CL, CR and each R_i are defined as below, where C denotes a class name, P a property name, P^- inverse of a property, $\exists R.\top$ an unqualified existential restriction and $\exists R.CR$ a qualified existential restriction (see [7]):

$CL ::= C \mid \exists R.\top$ $\quad\quad R_i ::= P \mid P^-$

$CR ::= C \mid \exists R.CR \mid CR \sqcap CR$

The semantics of a KB is the usual interpretation-based semantics in description logics [7]. Note that here we focus on the essential parts of DL-Lite$_R$; those axioms that actually participate in the query rewriting algorithms and are captured by TGDs. In particular, we do not consider negative inclusions; these are commonly treated using a pre-processing phase [17] where an algorithm checks that they are satisfiable and they are subsequently ignored for further processing. As well, note that sometimes DL-Lite$_R$ is presented without intersection or qualified existential restriction in the right-hand side but these can be freely added [16].

Query answering in data integration and exchange is defined using the notion of *certain answers* [25]. Given a source instance I, a set of source-to-target dependencies (or mappings) Σ_{st}, and a set of target dependencies (or axioms) Σ_t, a *solution* for $<I, \Sigma_{st}, \Sigma_t>$ is an instance J such that $<I, J> \models \Sigma_{st}$ and $J \models \Sigma_t$. Given $<I, \Sigma_{st}, \Sigma_t>$ as above, a tuple of constants $\vec{c}$ and a CQ q, $\vec{c}$ is a *certain answer* of q if (i) $\vec{c} \in q(J)$ for all solutions J for $<I, \Sigma_{st}, \Sigma_t>$, and (ii) $\vec{c}$ contains only constants that appear in I, Σ_{st}, and Σ_t (that is, no *labelled nulls* as we will see below). Forward and backward-chaining are the two prevalent approaches to obtain certain answers.

2.2 Forward-chaining

Forward-chaining algorithms compute the closure of a given database or triplestore by an exhaustive, bottom-up (i.e., starting from data), application of the rules in a forward manner (i.e., matching the premise and inferring the consequent for every rule). The chase family of algorithms [1, 12] is employed to "transfer" data from the sources to a target centralised warehouse and at the same materialise in this warehouse all entailed inferences from the target dependencies/ontology. Certain answers can be computed by running the query only on this centralized data warehouse, defined as the *universal solution* [25]. Several dedicated systems have been developed for forward chaining during the last few years [3, 14, 18, 28, 36, 47, 63].

In this paper, we focus on the *standard chase* algorithm, referred to simply as the chase. The choice of the particular chase flavour is orthogonal to our benchmark, scenarios, systems and experiments and not a restriction of any kind. In other words, to run another chase system (e.g, implementing the skolem chase variant) we just plug it into the benchmark by a call to the relevant system; datasets and tools (e.g., parsers, translators) or the experimental workflow are unaffected. The chase algorithm is a sequence of repeated *chase steps*. Every step starts from an st-TGD, detects a homomorphism from the body of the st-TGD to the source instance and creates tuples according to the head of the st-TGD to be inserted in the target instance. The target instance is further chased under the target dependencies and the whole process of running chase steps is repeated exhaustively to create the universal solution [12].

Without considering the full details of the chase algorithm, we will illustrate it with an example, modified from [25] and shown in Table 1. The table shows a data exchange setting with one source relation $DeptEmp$ that stores department ids, employee names and ids, and three target relations: $Dept$ with the id of each department and its manager, Emp with ids of employees and the departments they work in, and $EmpiInfo$ with employee ids and names. In Table 1, one can also see the DL-Lite counterpart of this setting which we discuss later. The chase starts by applying the source-to-target dependencies Σ_{st} to the source instance I, which produces an instance $J = I \cup \{Dept(IT, m_1), Emp(E003, IT), EmpInfo(E003, Mary)\}$ where m_1 is a "witness value" or labelled null, i.e., a value inserted in the place of an existential variable [25]. Note that the two GAV dependencies in Σ_{st} are subsumed by the single LAV one and they don't produce additional facts. This instance does not satisfy the target constraints and more steps of the chase with Σ_t are needed, which produce an updated instance $J_2 = J \cup \{Emp(m_1, IT), EmpInfo(m_1, n_1)\}$, where n_1 is another labelled null. Since J_2 satisfies all constrains Σ_{st} and Σ_t, it is a universal solution and can be used to obtain certain query answers; for the query shown in the table the only certain answer tuple is $(IT, Mary)$.

2.3 Backward-chaining

In contrast to computing an inference closure, backward-chaining approach is a top-down approach which starts from the query and processes the ontology axioms or rules in a 'backward' fashion. By processing from consequent to premise, backward-chaining generates a query-rewriting that incorporates ontology entailments. In recent years, a number of relevant algorithms/systems have been developed [8, 20, 31, 50, 56, 64], and much effort has been given into producing small rewritings that can be efficiently executed.

Most query rewriting algorithms have been designed to consider ontologies in DL-Lite$_R$. Conjunctive query answering over DL-Lite$_R$ KBs is also using the certain answer semantics; however, this is commonly achieved by query rewriting [17]. Note that, while chase systems can support arbitrary complex queries (simply run an SQL query on top of a universal solution) query rewriting using more expressive queries than CQs is a domain of active research and not supported by most OBDA systems. OBDA algorithms rewrite the query Q to a rewriting Q', usually a union of CQs (UCQ) or a datalog query, to be executed over $\mathcal{A}$. In some cases, $\mathcal{A}$ can be queried directly to obtain certain answers. However, often there is no explicit A-Box, as in our example in Table 1 but instead systems might use a mapping language like R2RML [22] or OBDA Mappings [9, 48], which maps the concepts and properties of $\mathcal{T}$ to data in an RDBMS or a triplestore. In these cases, there is a second "rewriting" phase that translates Q' into an SQL or SPARQL query using the mappings. Table 1 shows the OBDA mappings corresponding to the GAV st-TGDs of the example. Note that mapping languages in OBDA only capture GAV dependencies and so the LAV rule is not used in this example. In fact, LAV in OBDA has not been specifically addressed [54, 58, 61].

The most representative query rewriting algorithm is *Perfect Reformulation* [17] and without giving the full details we are intuitively explaining it here using our example of Table 1. The algorithm starts from the query atoms of Q. It unifies a query atom in each step with an atom on the right-hand side of an axiom of $\mathcal{T}$ (or the head of the corresponding TGD in the data exchange version), and replaces this appropriately with the atom on left of the

Table 1: DL-Lite/TGD mapping example

		DL-Lite/OBDA-mapping	TGDs
Target Schema (T)		$Dept(dID, mID), Emp(eID, dID), EmpInfo(eID, eName)$	
Source Schema (S)		$DeptEmp(dID, eName, eID)$	
T-**Box** ($\mathcal{T}$) Σ_t		$Dept \sqsubseteq Emp^-$ $Emp \sqsubseteq \exists EmpInfo.\top$	$Dept(d, m) \rightarrow Emp(m, d)$ $Emp(e, d) \rightarrow \exists\ nEmpInfo(e, n)$
Mappings Σ_{st}	LAV	N/A	$DeptEmp(d, n, e) \rightarrow \exists m\ Dept(d, m) \wedge Emp(e, d) \wedge EmpInfo(e, n)$
	GAV	target ns:ns/$\{eID\}$ ns:Emp ns:ns/$\{dID\}$ source SELECT eID, dID FROM $DeptEmp$	$DeptEmp(d, n, e) \rightarrow Emp(e, d)$
		target ns:ns/$\{eID\}$ ns:EmpInfo ns:ns/$\{eName\}$ source SELECT eID, $eName$ FROM $DeptEmp$	$DeptEmp(d, n, e) \rightarrow EmpInfo(e, n)$
Query		$Q(d, n) \leftarrow Emp(e, d), EmpInfo(e, n)$	
Source Instance		$I = \{DeptEmp(IT, Mary, E003)\}$	
A-**Box** ($\mathcal{A}$)		$\emptyset$	

axiom (the body atom). This creates a new query with the replaced atom. It then continues exhaustively by choosing the original query or one of its rewritings and replacing a subsequent atom. We will refer to the (usually UCQ) output of the algorithm as the *rewriting* and to the individual conjunctive queries that are its elements as the *conjunctive rewritings*. In Table 1, the first conjunctive rewriting Q_1 will be the query itself. Next, the algorithm matches *Emp* in the query with the first axiom in $\mathcal{T}$, appropriately substituting *Emp* with the body of this rule which is *Dept*, resulting in Q_2.

$$Q_1(X, Y) \leftarrow Emp(Z, X), EmpInfo(Z, Y)$$
$$Q_2(X, Y) \leftarrow Dept(X, Z), EmpInfo(Z, Y)$$

There is no other conjunctive rewriting that the algorithm can produce. Intuitively, if we try to rewrite $EmpInfo(Z, Y)$ in any query with the second axiom from $\mathcal{T}$, it will end-up in a query that does not contain Y, which is a free variable needed by the query (more details in [17]). The final rewriting containing union of Q_1 and Q_2 could be executed over the A-Box or if the latter is virtual, as in our example, use OBDA-mappings to answer it. In Table 1, *Dept* and *EmpInfo* are mapped to the database table *DeptEmp*; in a last step we unfold the conjunctive rewritings, creating SQL queries. E.g., for Q_2 we get: *SELECT A.dID, B.eName FROM DeptEmp AS A , DeptEmp AS B WHERE A.eID = B.eID*. The SQL for Q_1 is similar and the algorithm executes their SQL UNION on the database (represented by I) to obtain the same answer as in Section 2.2.

3 FROM DL-LITE$_R$ TO LTGDS

In this section we describe an exact, previously unknown, correspondence between a restriction of LTGDs and DL-Lite$_R$, as defined in Section 2. The usual translation between these languages is "loose"; it is known that LTGDs capture most of DL-Lite$_R$. However not all LTGDs are translatable to DL-Lite$_R$; next, we precisely characterise the part of LTGDs that is. Note that, our criterion captures DL-Lite$_R$ without caring for the particular syntactic variation - although our tool that translates a set of TGDs to DL-Lite$_R$ uses the (OWL version of the) variant defined in Section 2.

We will make use of the notion of Predicate-Join patterns (PJs), which are graph representations of TGDs defined in [36] as follows. Let σ be a single TGD over schema R, where every relation schema has arity of at most two. The *PJs-graph for* σ is a graph that consists of a set of predicate nodes, variable nodes, and edges. For every atom in σ there is a predicate node labeled with the particular predicate name. Edges connect predicate nodes to variables nodes; the latter represent the variables of an atom with this predicate name. Edges are labeled with the argument position that the corresponding variable has inside the atom. Variable nodes are constructed as follows: (1) every variable that appears in the body of σ corresponds to a universal variable node, represented by the symbol $\bigcirc$; and (2) every variable that appears *only* in the head corresponds to an existential variable node, represented by $\otimes$. Figure 1 shows PJs-graphs for two example LTGDs. A universal variable that is common to the body and the head of σ is called a *frontier variable*.

We are now ready to define the language of linear constraints that are translatable to DL-Lite$_R$, called *PJ-acyclic TGDs*. We remind the reader that a graph is *connected* when there is a path from every node in the graph to every other node.

Definition 1. *For all LTGDs σ, that use relations of at most arity two, σ is a PJ-acyclic TGD if the PJs-graph for σ (1) is connected, (2) does not contain cycles between exactly two nodes, and (3) does not contain cycles that include existential variable nodes.*

Figure 1(a) shows a non PJ-acyclic TGD, where there is a join between atoms $R2$ and $R3$ on the existential variable W, creating a cycle that contains W. If W was a universal variable then this TGD would be a PJ-acyclic TGD. For example, Figure 1(b), although containing cycles, is still PJ-acyclic and translatable to DL-Lite$_R$.

Intuitively, the first condition forbids arbitrary (existential) cross products in the head of the TGD or the case when the head and the body are disconnected. The second condition forbids joins within the same atom such as $P(X, X)$, not modeled by DL-Lite. Condition (3) of Def. 1 ensures that a TGD is translatable to an inclusion axiom.

When there is a unary atom in the body of a TGD this translates to a subclass axiom. When the body atom is binary this might translate to a domain or range axiom, or both (if both universal variables are frontiers), or a subproperty axiom if both universal variables are frontiers and are "passed" to the same head atom. In fact, these are the only cases where we can have two frontier variables. In particular, a TGD cannot contain two frontier variables and the two "connected components" in the head that contain these variables to be joined existentially; it is in this sense that the graph of a translatable TGD should be acyclic.

Note that, one could use a definition based on hypergraphs [30] or some variation [39] to define our PJ-acyclic notion. However, this also would not be immediate; the hypergraphs for $A(X) \rightarrow C(Y)$ or $P(X, X)$ would be acyclic but not translatable to DL-Lite$_R$. We

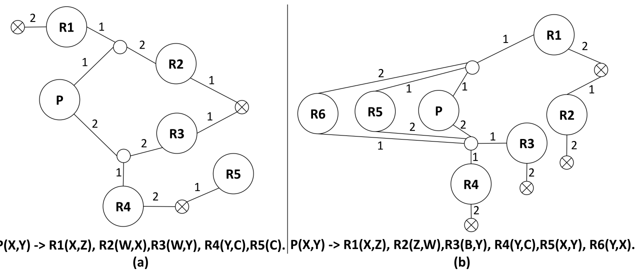
R1
R2
R3
R4
R5
R6
P
P(X,Y) -> R1(X,Z), R2(W,X),R3(W,Y), R4(Y,C),R5(C).
(a)
P(X,Y) -> R1(X,Z), R2(Z,W),R3(B,Y), R4(Y,C),R5(X,Y), R6(Y,X).
(b)

Figure 1: PJs-graphs; (a) non PJ-acyclic TGDs: cycle on existential variable and (b) PJ-acyclic TGDs: no cycle on existential variable

elected to work with the PJ notation that treats multiple occurrences of the same variable separately (e.g., in $P(X, X)$) .

To give our formal correspondence between DL-Lite and TGDs, we need to define the notion of semantic equivalence. For convenience, we consider an *A-box database*: that is, an *A*-box *O* that is backed up by a relational database containing exactly one atomic relation for every concept of *O*, populated with a tuple for every instance of the concept, and one binary relation for every role, which again, is populated with tuples for every property assertion in *O*.

Definition 2. *Given an A-box database O, a TGD σ, and a DL-Lite axiom μ, σ is semantically equivalent to μ, iff σ is satisfied on O whenever μ is satisfied on O.*

Theorem 1. *For every PJ-acyclic TGD there is a semantically equivalent DL-Lite$_R$ axiom and vice versa.*

Next, we outline the translation between PJ-acyclic TGDs and back; through this translation the correctness of the theorem becomes apparent. One direction, from DL-Lite$_R$ to TGDs, is straightforward and corresponds to the logic-based semantics of DL-Lite$_R$. This correspondence can be seen for some example axioms in Table 2, and is also described (as part of another algorithm) in Algorithm 1. A part of the other direction, in particular, to translate the body atom of a TGD to the left-hand side of an inclusion axiom, is relatively straightforward as well. In fact, Table 2 shows all different cases of how body atoms are translated to left-hand sides of inclusion axioms depending on which variables of the body are frontier.

For translating the head of the TGD our algorithm considers, and translates in isolation, different components of the head that are maximal w.r.t. set inclusion and *existentially connected*: either the component is an atom with only frontier variables, or all atoms in the component share an existential variable with some other atom in the component. This is without loss of generality as we can always decompose a TGD by "breaking" the head on components that share only universal variables resulting to an equivalent set of TGDs that have the desired property. Thus, for every PJ-acyclic TGD we conceptualise a TGD with the same body and a different existentially connected head component. We then translate the body atoms as mentioned. To translate head atoms which have only frontier variables we can refer to Table 2. To translate head components which contain existential variables, we start from the binary atom that contains the frontier variable *u* (there has to be one due to PJ-acyclicity); let *P* be this atom and *z* its existential

Table 2: Conversions between Description Logic and TGDs

Description Logic	TGD Rule
$A \sqsubseteq B$	$A(x) \rightarrow B(x)$
$R \sqsubseteq P$	$R(x, y) \rightarrow P(x, y)$
$R \sqsubseteq P^-$	$R(x, y) \rightarrow P(y, x)$
$A \sqsubseteq \exists P.\top$	$A(x) \rightarrow P(x, y)$
$A \sqsubseteq \exists P^-.\top$	$A(x) \rightarrow P(y, x)$
$\exists R.\top \sqsubseteq B$	$R(x, y) \rightarrow B(x)$
$\exists R.\top \sqsubseteq \exists P.\top$	$R(x, y) \rightarrow P(x, z)$
$\exists R.\top \sqsubseteq \exists P^-.\top$	$R(x, y) \rightarrow P(z, x)$
$\exists R^-.\top \sqsubseteq \exists P.\top$	$R(x, y) \rightarrow P(y, z)$
$\exists R^-.\top \sqsubseteq \exists P^-.\top$	$R(x, y) \rightarrow P(z, y)$
$A \sqsubseteq \exists P.P_1$	$A(x) \rightarrow P(x, y), P_1(y)$
$\exists R.\top \sqsubseteq \exists P.(\exists P2.(\exists P3.D \sqcap C))$	$R(x, y) \rightarrow P(x, z), P_2(z, w), P_3(w, g), C(w), D(g)$

variable. If *u* is in $P's$ first position, this will form the beginning of an existential restriction expression of the form "$\exists P.\Phi$" using a role name *P*. If *u* is in the second position, this will translate into using an inverse role expression, e.g., "$\exists P^-.\Phi$". In both cases, sub-expression Φ will be recursively unfolded based on the rest of the joins in the TGD. When, for example, another binary atom P_2 contains *z* this will also be translated, within Φ, to "$\exists P_2.$" or "$\exists P_2^-.$" depending on where *z* lies in it.

```
input    : A PJ-acyclic TGD $\sigma$ with an existential
           component in the right-hand side
output   : String $\alpha$: the DL-Lite translation of $\sigma$
function : translateExResTGD($\sigma$)
1  Initialize String $\alpha$ := “”
2  if body atom A of $\sigma$ is unary then
3      $\alpha$ := ”A”
4  else if body atom R of $\sigma$ is binary then
5      if the frontier variable of $\sigma$ is in the first position of R
        then
6          $\alpha$ := ”$\exists R.\top$”
7      else if the frontier variable of $\sigma$ is in the second position
        of R then
8          $\alpha$ := ”$\exists R^-.\top$”
9  $\alpha$ += ” $\sqsubseteq$ ”
10 headUAtom := the head atom of $\sigma$ that contains the
   universal variable, u
11 return $\alpha$ + translateRightSide(headUAtom,u, $\sigma$)
```
Algorithm 1: TGD with existential component to DL-Lite

Algorithm 1 translates a TGD with a single existential component in the head by translating the left hand-side (i.e., body) of the TGD and then calling the recursive function of Algorithm 2 to translate all joined atoms in the right hand-side. ForBackBench implements these algorithms and offers tools for PJ-acyclic TGD recognition and conversion back and forth to DL-Lite$_R$.

4 EXISTING BENCHMARKING APPROACHES

In this paper, we present the first benchmarking framework for cross-approach comparisons between forward- and backward chaining implementations. Nevertheless, just in the OBDA space alone, while there are many comparisons of rewriting systems, there is

```
input    : An atom, $\tau$, to be translated together with its
           existentially connected component in $\sigma$, to an
           existential restriction; a variable $u$ the position of
           which in $\tau$ determines the role/inverse role use;
           the entire TGD $\sigma$
output   : String $\alpha$: the DL-Lite translation of $\tau$ and its
           existentially connected component in $\sigma$
function : translateRightSide($\tau$, $u$, $\sigma$)
Initialize String $\alpha$ := “”
if $\tau$ is a unary atom $A(x)$ then
    $\alpha$ = “$A$”
else if $\tau$ is a binary atom $P(x, y)$ then
    if $u$ is in the first position of $\tau$ then
        $\alpha$ += ”$\exists P$.”
    else if $u$ is in the second position of $\tau$ then
        $\alpha$ += ”$\exists P^-$.”
    $\alpha$ += ”(”
    Initialize $z$ := the other variable of $\tau$
    foreach atom $\rho$, other than $\tau$, in $\sigma$ that contains $z$ do
        $\alpha$ += TranslateRightSide($\rho$, $z$, $\sigma$)
        remove last occurrence of ”.” and ”(” from $\alpha$
        $\alpha$ += ” $\sqcap$ ”
    remove last occurrence of ” $\sqcap$ ” form $\alpha$
    $\alpha$ += ”)”
return $\alpha$
```

Algorithm 2: TGD head to an existential restriction

only a handful of benchmarking efforts; to the best of our knowledge, ours can be seen as a holistic and comprehensive OBDA benchmark that aggregates all pre-existing scenarios over systems and implementations of different granularities, supports different kinds of mappings by pairing rewriting algorithms with view-based systems and/or a database back-ends, and streamlines the process for future experiments and unforeseen systems, providing an automated testing environment that is easily reusable or extensible.

Indeed, for comparing query-rewriting systems, the Norwegian Petroleum Directorate (NPD) Benchmark [38] provides the most recent benchmark platform, which supports automated testing and data generating. The NPD Benchmark tested only two query-rewriting systems, Ontop [56] and Stardog [21]. Additionally, NPD has to be used with complete end-to-end OBDA systems that have a supported data back-end, use R2RML or OBDA-mappings and have implemented a compatible API to NPD. This is in contrast to our framework in which one can simply plug in an algorithm implementation solving a part of query answering (e.g., just the rewriting). Another comprehensive OBDA benchmark, which unfortunately is not publicly available, is provided by Mora and Corcho [43] which tests popular and well-researched backward-chaining systems such as Requiem [50], Rapid [20], and Nyaya [31]. Mora and Corcho also identify the main example scenarios used in previous evaluations: the Adolena, StockExchange, University, and Vicodi benchmark scenarios. These scenarios were all first introduced by Perez-Urbina et al. [49] and have since been used extensively in subsequent work on comparing backward-chaining algorithms [20, 33, 57]. Efforts that focus simply on providing datasets and not entire experimental frameworks, include the Texas Benchmark [59] with a range of scenarios, or approaches to extend existing scenarios such as Extended-LUBM [42], which modified the LUBM scenario [32] to support OBDA mappings. In the broader Semantic Web area, benchmarking has focused on comparing RDF triplestores for query performance, such as in FishMark [10], BSBM [13], and DBpedia [44].

In the space of forward-chaining approaches, STBenchmark [5] demonstrates an early approach towards the benchmarking of mapping systems. Although the tools are no longer available, one of the future challenges that [5] mentions is the need for a uniform testbed for data exchange tasks. The chaseBench suite [12] provides a comprehensive evaluation of seven chase-based systems over a wide range of scenarios. Likewise, it also raises the future challenge of accounting for and benchmarking alternative approaches to query answering. ForBackBench builds upon the existing chaseBench framework [12], extending it to incorporate backward- as well as forward-chaining. iBench [6], while not a benchmark system itself, is a metadata generator that can be used in order to supplement other approaches through the generation of common integration tasks and scenarios (such as mapping creation or schema evolution) at configurable sizes and using controllable constraints.

5 FORBACKBENCH

ForBackBench consolidates seven data integration/exchange scenarios with eight query answering systems to provide a streamlined automated means for cross-method benchmarking, which also allows for easy future extension. The ForBackBench framework is made available as an open-source project on GitHub (https://github.com/georgeKon/ForBackBench). Our converters and tools are available to be used also as standalone programs.

5.1 Scenarios and Data

We incorporate a number of existing high-quality benchmarking scenarios (see Table 3), including the four scenarios of Pérez-Urbina et al. discussed earlier [49]. Surprisingly, although most of these scenarios were developed to benchmark query rewriting, they are all chase-terminating with the exception of Adolena; to use this we remove one axiom from Adolena's dependencies that is causing an infinite chase. In addition, we include OWL2Bench [60]; this benchmark is designed to provide wide coverage of the OWL 2 language, and has the ability to test scalability by providing arbitrarily large ontologies. We also use the Deep scenario [12] from the chaseBench framework originally developed in [36]. To transform arbitrary DE scenarios to DL-Lite$_R$ we have implemented a tool that prunes the arities of target schemas down to binary and does this as well for the target dependencies and the heads of the source-to-target mappings. Most scenarios include at least five queries, and for those that don't, we have manually created queries in order to have five for each scenario. Moreover, most of these scenarios come with no data; as such, we have developed a data generator to serve each scenario. We generate data with four different data sizes, that we call: *small*, with up to 1000 facts for each table; *medium*, with up to 10000 tuples; *large*, with up to $100,000$ tuples; and *very large*, with up to $1,000,000$ tuples. Furthermore, we also integrate scenarios

Table 3: List of included data scenarios

Scenario	Source	Original Format	Data
Adolena	[49]	DL-Lite	Generated
Deep	[12]	chaseBench	Generated
NPD	[38]	DL-Lite	Included
OWL2Bench	[60]	DL-Lite	Generated
StockExchange	[49]	DL-Lite	Generated
University	[49]	chaseBench	Generated
Vicodi	[49]	DL-Lite	Generated

that include real-world data, such as the NPD [38]. This scenario is composed of a database schema and a data generator. We use NPD's data generator to generate tuples in tables which range between three to eighty columns. Some of these tables are small (i.e containing only 3 tuples) and other tables are large (i.e containing over 27, 000 tuples). Our data generation creates databases from a few hundred kilobytes for our small scenarios, designed to mimic running the systems on small structures like formulas or queries, to a few gigabytes for our "very large" scenarios where we might wait for tens of minutes for some systems to run on commodity hardware. This scale seems to be enough to show the tendencies and differences of the system's performance as the data grows. For larger experiments one should be able to run our benchmark out of the box on a larger server. As well, ForBackBench is easily extensible; it is simple to include additional scenarios, either in OWL/DL-Lite or in TGDs, as necessary with minor modifications.

5.2 Query-Answering Systems

As pointed out in Section 4 ForBackBench can serve as a comprehensive benchmark for OBDA alone (the same way chaseBench is the reference benchmark for chase implementations). Thus we have included all query rewriting systems freely obtainable online. These, (by nature) are memory-based. For chase-based systems we include a memory-based and a disk-based system from ChaseBench (RDFox [12] and Llunatic [28] respectively) as well as a newer memory-based system (Rulewerk [63]).

Three of the backward-chaining systems, Rapid [20], Iqaros [64], and Ontop [56], were chosen as established systems in the field that have performed very well in recent literature [33, 43, 64]. We also use OntopR, which is the internal T-Box query rewriting system of Ontop and have also included Graal [8] which is a research prototype. Alongside these tools, the framework also integrates the ChaseGQR system [36], which combines features from both forward- and backward-chaining approaches. In particular ChaseGQR runs forward-chaining-like preprocessing on the source mappings and then relies on GQR[35] to run a "query answering using views" algorithm in order to produce a source rewriting.

5.3 Supporting Data Integration

One of our objectives is to validate query answering tools and scenarios in a data integration/exchange setting, and in particular in the face of source-to-target mappings. Note that, we use the term Data Integration in the sense first used in database research [23, 24, 29, 40] related to the problem of answering queries over different schemas using schema mappings or views. Broader notions of data integration include problems such as transforming data between different formats such as CSV, XML, and JSON and there have been established benchmarks in this space [52].

Table 4: Benchmarked systems & Query Answering process

DI System	One-to-One	GAV	LAV
Rapid	Native	Unfolding	GQR
Iqaros	Native	Unfolding	GQR
Graal	Native	Unfolding	GQR
Ontop	Native	Native	
OntopR	Native	Unfolding	GQR
RDFox	Native	Native	Native
Rulewerk	Native & RST	Native	Native & RST
ChaseGQR	Native		Native
GQR	RST		RST
Llunatic	Native	Native	Native

Most DL-Lite scenarios do not use source-to-target mappings except for NPD, which uses OBDA-mappings. We implemented translators between OBDA-mappings and TGDs, and mapping generators that extend the original scenarios to include generated LAV and GAV mappings. We also generate simple one-to-one mappings that associate each ontology concept or role to a unary or binary database relation. These 1-to-1 mappings of T-box predicates to database relations can be seen as an A-box implementation; i.e., as a "native" support of a database connection for DL-Lite systems, which most of them do not have. Ontop is an exception which is a complete end-to-end system with OBDA-mappings (that are GAV).

One-to-one and GAV mappings are in principle simple to use; we can "unfold" [46] a query or a query rewriting using the mapping definitions. Indeed, we implemented an unfolding algorithm for OBDA systems. This is trivial for 1-to-1 mappings and hence we refer this support as "native" for all OBDA systems, while we reserve the term "unfolding" for the additional GAV support. For LAV scenarios for OBDA systems, rewriting the T-box rewritings over the mappings to obtain the final source queries is not trivial; in effect there are dedicated algorithms for this problem, known as *query answering using views* and we use the GQR algorithm [35] which is the state-of-the-art.

Lastly, we also consider an approach from [2], where the authors study a reduction of a DE setting $<\Sigma_{st}, \Sigma_t, I, Q>$, with linear t-TGDs, to a setting $<\Sigma'_{st}, I, Q>$, with no t-TGDs. This reduction is performed by running the chase on the st-TGDs using the t-TGDs and producing longer st-TGDs (which however are always (G)LAV). The authors prove that the new DE setting produces the same certain answers as the original setting. We implemented this algorithm and named it "RST"; we used Rulewerk [18] to chase the st-TGD mappings since it tends to perform the fastest on small databases. The reduced setting $< \Sigma'_{st}, I, Q >$ can be treated with any chase algorithm (we tested with a second use of Rulewerk) or with GQR [35] to obtain the certain answers. We run the RST on both LAV and the one-to-one mappings of our scenarios, creating in both cases longer LAV mappings. We did not run RST directly on GAV mappings as this would create GLAV mappings, which would then need to be broken down to LAV and GAV [27].

Table 4 shows a summary of the benchmarked systems and the support we have implemented for different mappings. For 1-to-1, most systems are "native", i.e., they run almost directly to answer

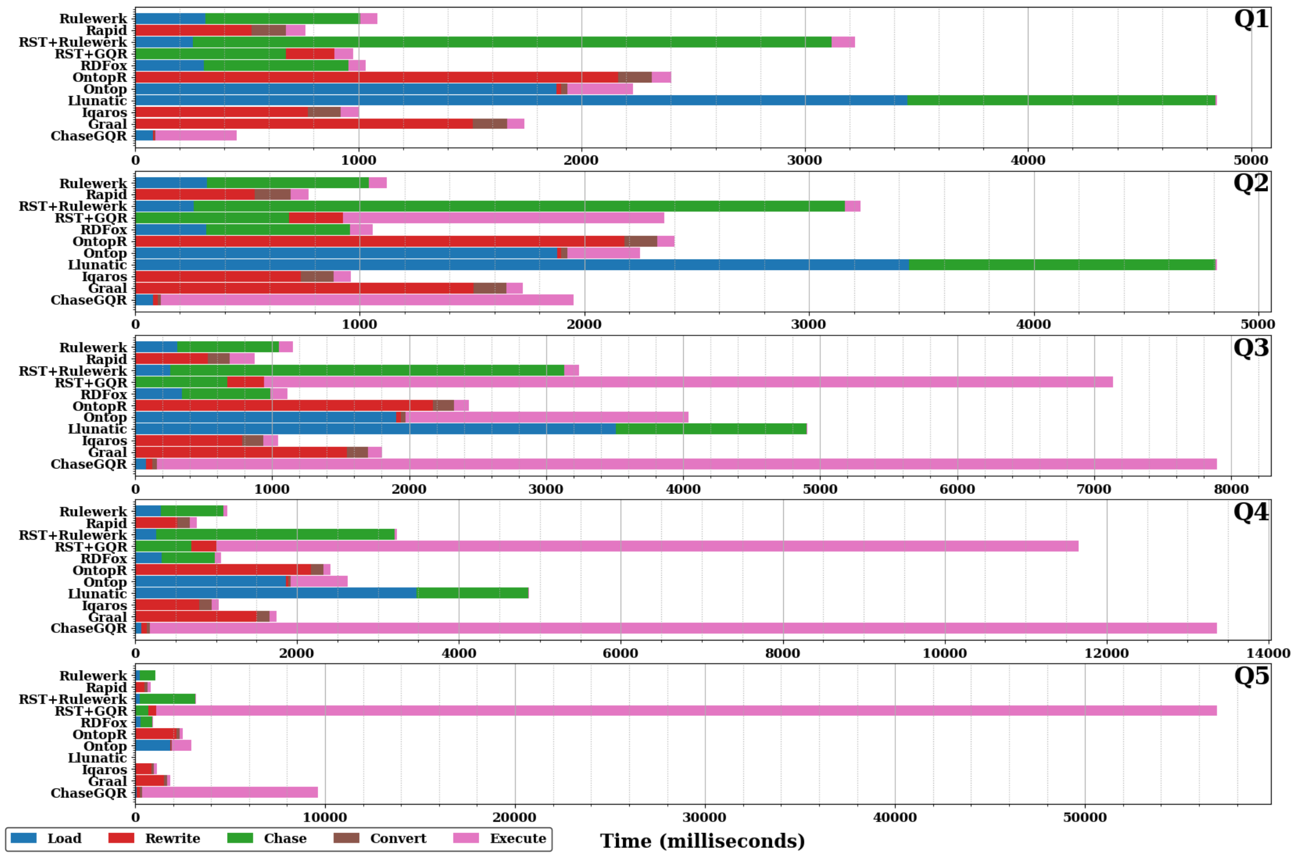

Figure 2: StockExchange scenario with Medium data size and one-to-one mappings

the query. However, in GAV and LAV, additional functionality ("unfolding" and "GQR", respectively) is used after running the systems to fully answer the query. As mentioned, in the 1-to-1 and LAV scenarios Rulewerk has two running settings: "native" and "RST".

Note that, while our one-to-one mappings simulate an A-Box in the database (and hence are also on a binary schema), our GAV and LAV mappings, mapping generators, and underlying databases are not constrained to binary source schemas. The SQL rewritings produced in these settings use uniformly distributed arities of between 2-5 attributes, for synthetically generated scenarios, while NPD contains some extremely wide relations with up to 78 attributes.

6 EVALUATION

Our experiments were run on an Ubuntu server with Intel(R) Xeon(R), 2.30GHz, 8-core processor, 32GB of memory, and a total of 512GB of disk space. Our GitHub includes an online appendix with extensive experiments and results; due to space limitations here we only present a few figures but base our discussion on all results.

6.1 Methodology

Each one of our benchmarking workflows was executed as a series of discrete stages: data generation (which was not timed), data loading into a database or main memory (which might be a different process for a different system) reported in our experiment graphs in blue, chasing (either data or mappings via RST) reported in green, rewriting queries using the T-box or target constraints reported in red, converting the rewritings to executable queries either via simple 1-1 conversion, unfolding for GAV or running GQR for LAV, all reported in brown, and finally execution of queries and/or query rewritings over the data, reported in pink.

Each workflow was run over the 7 scenarios discussed in Section 5.1, using one-to-one, LAV, or GAV mappings, with small, medium, large, and very large data stores; this was done for all systems depending on their setting using our discussed converters.

For each scenario, five queries of different sizes were executed. The size of queries (also summarised in Table 8 in our online appendix) varies from one to seven atoms. Each system and query were run fresh from end-to-end six times, with the first time discarded as a 'cold-run'. Each stage of the workflow had a maximum permitted runtime of 30 minutes; any run exceeding this or stopping with an error was deemed a failure and results (including any previous successful stages, e.g. data loading) were not reported.

6.2 Results

6.2.1 One-to-One Mappings. Our first finding is that throughout the small datasets in experiments and independent of the mapping

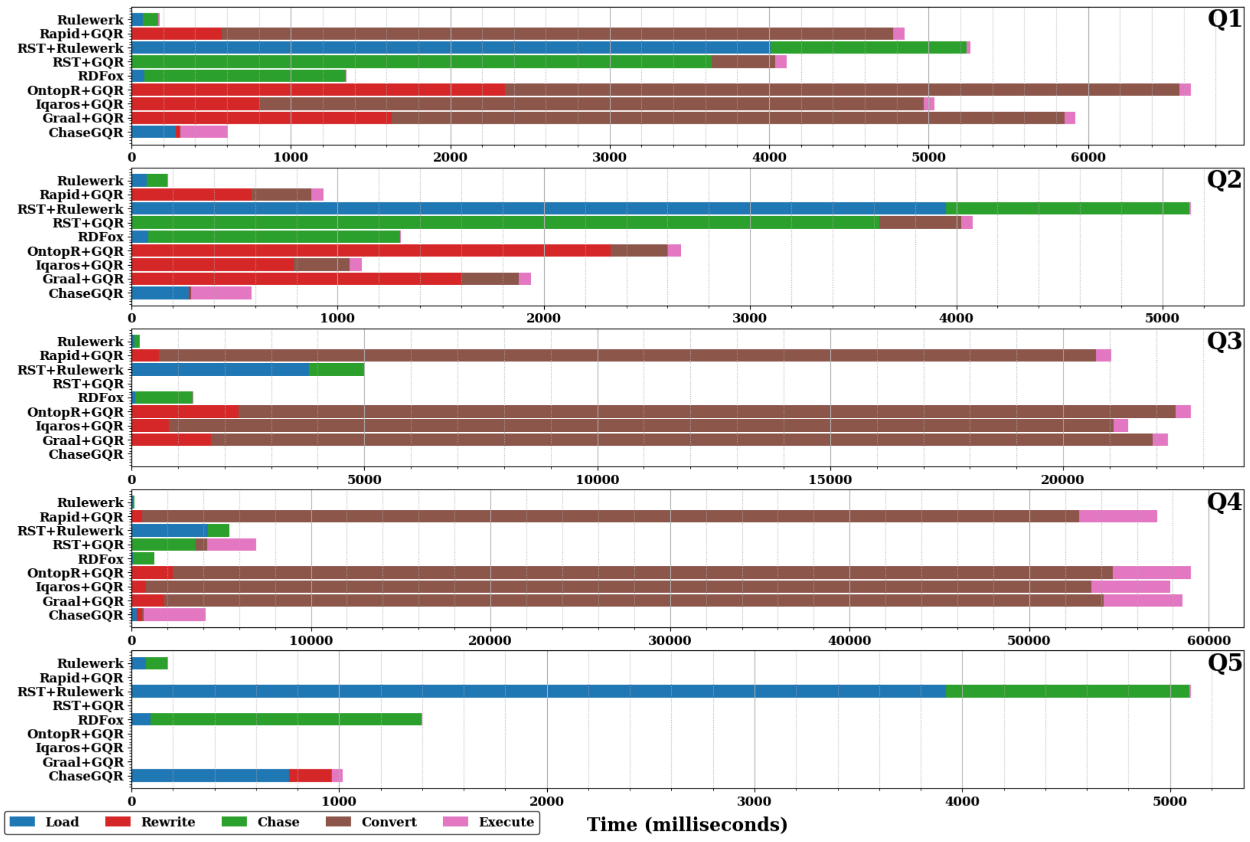

Figure 3: Vicodi scenario with Small data size and LAV mappings

language used, chase systems Rulewerk and RDFox indeed perform well, and alongside ChaseGQR are the fastest, when measuring the end-to-end time. Disk-based chase systems, in particular Llunatic, seems to be an overkill for small data. For 1-to-1 mappings, and medium and larger-sized datasets such as the StockExchange scenario shown in Figure 2, chase performance becomes comparatively slower, and is outclassed by query-rewritng systems such as Rapid. This figure also shows that Ontop, while requiring substantial time to complete its preprocessing step to build offline indices, has an execution time considerably faster than the majority of other approaches — something also apparent in the other scenarios. In addition, ChaseGQR's performance on datasets of this size is highly dependant on the number of the joins in the query as ChaseGQR uses GQR as an internal view-based query rewriting algorithm which has has been shown to have variable performance depending on the query size, and seems an overkill for 1-to-1 mappings [35].

For very large datasets, the execution time dominated the other stages. At this size, relative system performance appeared to be highly variable and dependant on each query. For example, Rapid performed amongst both fastest and slowest systems in the same scenario, depending on the query executed, as did Rulewerk (cf Figure 20 and Figure 21 in our git repo). Moreover, at this size, chase systems might exceed the time limit, e.g., for Vicodi and Deep.

Overall, comparing chase systems we can see Rulewerk is faster than RDFox on small and very large datasets while RDFox performs faster than Rulewerk on medium and large datasets (note that we have used the 2017 version of RDFox found in chaseBench). Llunatic is still considerably slower than memory-based systems. For query-rewriting systems, when measuring end-to-end time, Rapid is the fastest on small/medium datasets and Iqaros performs on par with Rapid in almost all large/very large scenarios.

6.2.2 LAV Mappings. Over small datasets, chase systems perform extremely well, while (particularly for queries Q1, Q3, and Q4 of the Vicodi scenario shown in Figure 3) the GQR rewriting phase proved challenging for many query-rewriting systems due to the more complex mappings. As we can see in the graph, LAV rewriting (so query answering using views) seems to be much slower than the T-box rewriting which is to be expected since it operates on exponentially large T-box rewritings. Nevertheless, this indicates the difficulty that current query rewriting systems have with LAV mappings. Other scenarios show similar results. This tendency shows even for the larger datasets; we can see that the assumption that as data grows query rewriting becomes faster is true only in some queries of the Deep scenario (Figure 32) where Rapid+GQR is faster than Rulewerk and RDFox. Indeed using LAV mappings

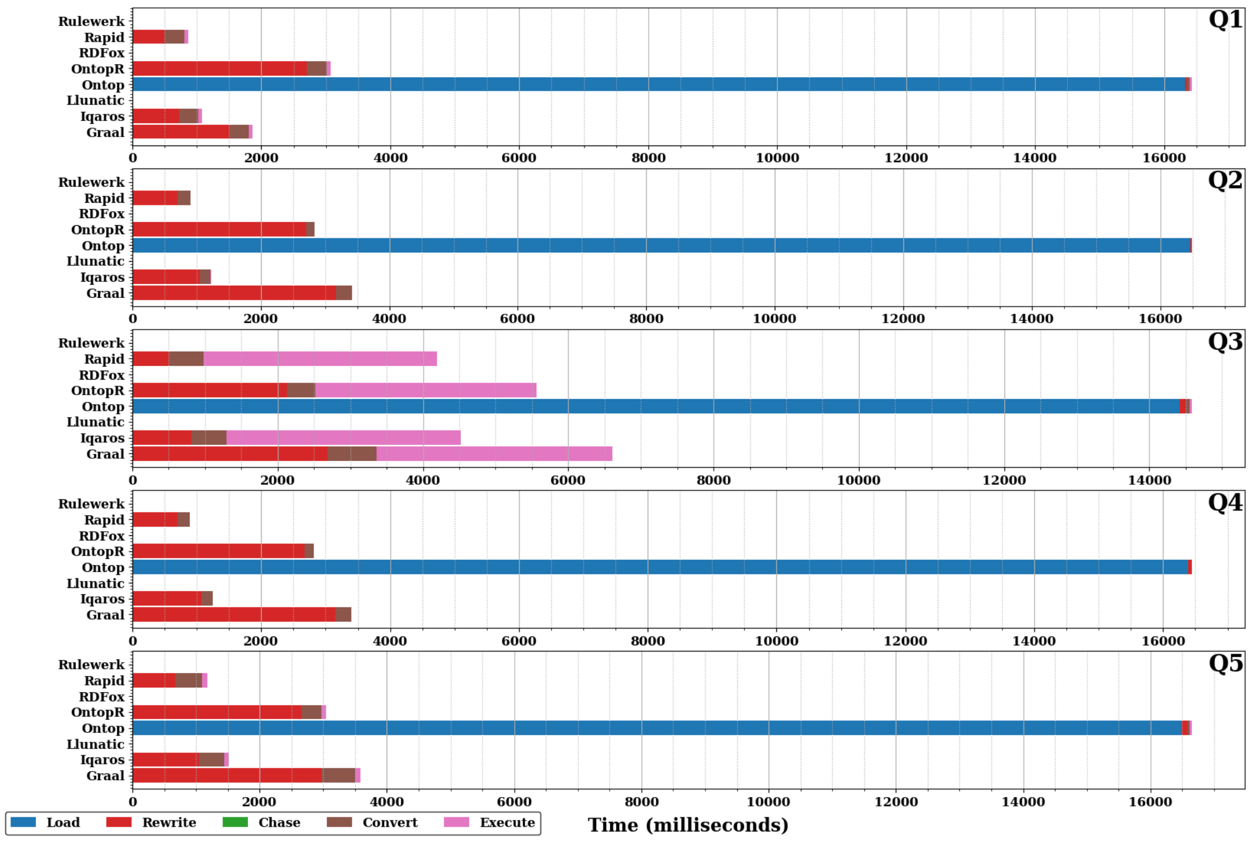

Figure 4: NPD scenario with Large data size, and GAV mappings

might be a primary factor for one to choose a forward chaining solution; at least until more optimised systems appear.

6.2.3 GAV Mappings. Whith small data, Rulewerk and RDFox performed well. And although again disk-based chase is slow, as shown in Figure 40 in the appendix, in some cases chase systems were the only ones to terminate. Even though the rewriting of the query itself is not dependent on the data, the execution time for OBDA systems dominates in the GAV case often exceeding the time limit (30 minutes) depending on the size of the rewritings as in Q5 in Figure 40. The execution time is relatively much slower when compared to the 1-to-1 case where most time goes into the T-box rewriting or to LAV view-based rewriting in the LAV case. As with 1-to-1 mappings, Ontop took substantial time to load data in its initial index, which could be dramatically reduced if it were allowed offline preprocessing (see Section 6.3). However, the execution phase of Ontop was very slow for some cases in (Figure 40 and Figure 42 in appendix).

This pattern was repeated for medium datasets (Figure 43 and Figure 44 in our git repo) and was also the case for most large datasets (Figure 45 and Figure 46 in our git repo). However, in the NPD scenario shown in Figure 4, which contains a large real-world dataset and simple GAV mappings, chase systems exceeded the time limit and failed to return results. While all query-rewriting systems performed consistently fast, Rapid performed substantially faster in this scenario, followed by Iqaros. This figure also highlights the contrast between the substantial load-time of Ontop and its dramatically shorter rewrite and execution times (orders of magnitude).

In very large datasets (around 1M tuples), the chase systems are very slow compared to query rewriting when the rewritings produced small unfolded queries. However, there are many cases, such as Q3 in Deep scenario shown in Figure 5, where query rewriting systems exceed the time limit because complex GAV mappings produce very large unfolded SQL queries.

The GAV mappings in NPD contain only one or two atoms in the body so our findings also suggest that query-rewriting systems perform well with simple GAV mappings and large data, whereas the chase systems perform best with complex GAV mappings (i.e., up to five atoms in the body) and smaller data.

Comparing chase systems across different kinds of mappings, Rulewerk is relatively more robust while RDFox suffers more from the generation of labelled nulls for existential variables, becoming relatively much slower for LAV mappings as compared to GAV.

6.2.4 Rewriting-complexity. In this section we perform an analysis of our results by classifying scenarios (in particular, query-ontology pairs) as easy, medium, or hard depending on the average number of conjunctive query rewritings that OBDA systems have on these scenarios. Table 5(a) shows the overall winner system on all scenarios

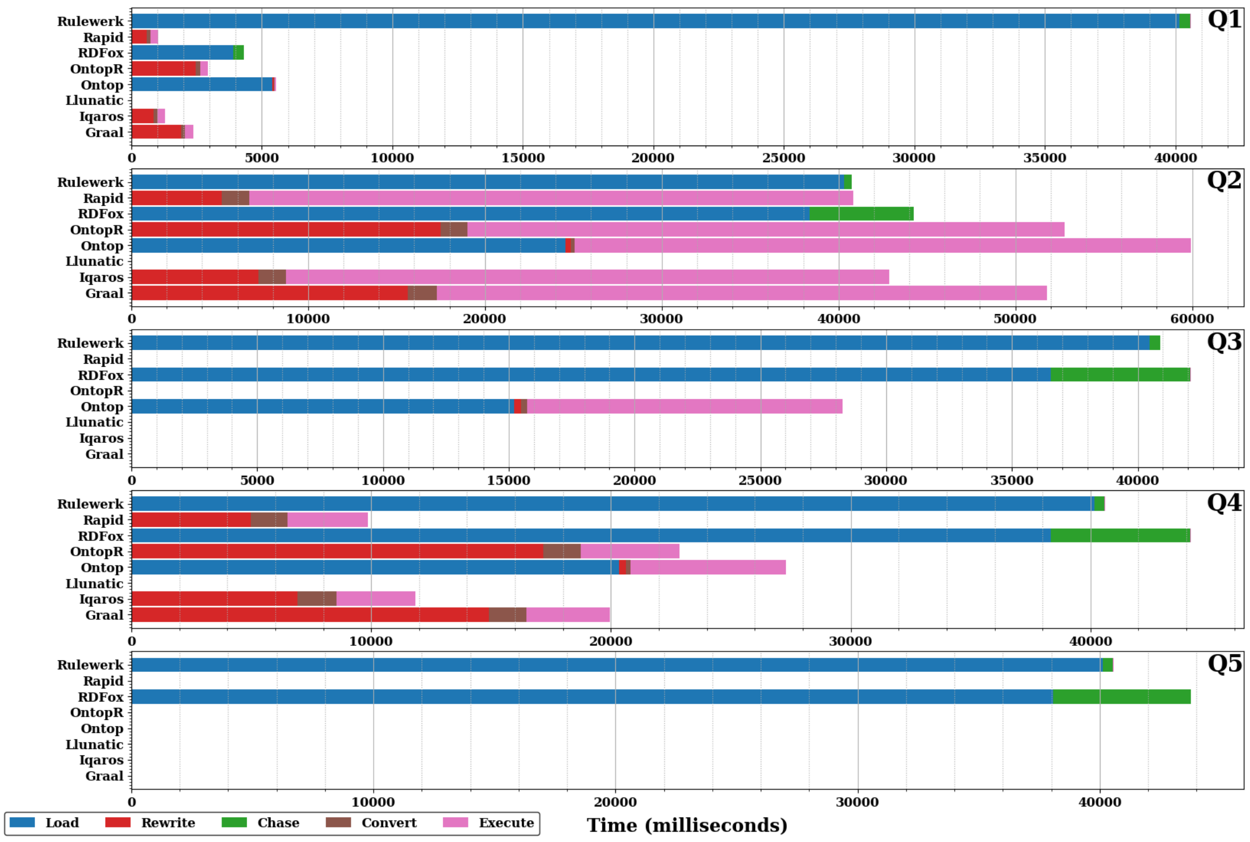

Figure 5: Deep scenario with Very Large data size and GAV mappings

Table 5: The state-of-art systems depending on number of rewritings per query

Query Cluster (# of Rewritings)	Avg. of # Rewritings	Global Winner	Small Data Winner	Medium Data Winner	Large Data Winner	Very Large Data Winner	One-to-One Mapping Winner	LAV Mapping Winner	GAV Mapping Winner
Easy	4	Rulewerk	Rulewerk	Rulewerk	Rapid	ChaseGQR	ChaseGQR	Rulewerk	Rulewerk
Medium	47	Rulewerk	Rulewerk	Rapid & ChaseGQR	Rapid	Rulewerk	Rulewerk	Rulewerk	Rulewerk
Hard	255	Rulewerk	Rulewerk	Rulewerk	RDFox	Rulewerk &Ontop	Rulewerk	Rulewerk	Rulewerk

(a) Comparing chase and query rewriting systems - end to end times

Query Cluster (# of Rewritings)	Avg. of # Rewritings	Global Winner	Small Data Winner	Medium Data Winner	Large Data Winner	Very Large Data Winner	One-to-One Mapping Winner	LAV Mapping Winner	GAV Mapping Winner
Easy	4	Rapid	ONTOP	Rapid	Rapid	ChaseGQR & Rapid	Rapid	Rapid	ONTOP
Medium	47	ONTOP	ONTOP & Rapid	ONTOP & Rapid	Rapid	ONTOP	ONTOP	ChaseGQR & Rapid	ONTOP & Rapid
Hard	255	ONTOP	ONTOP	ONTOP	ONTOP	ONTOP	ONTOP	ChaseGQR	OntopR &Rapid

(b) Compare query rewriting systems - without preprocessing times

of each category broken down in terms either of data size or mapping language. The results are very surprising; chase (in particular Rulewerk) seems to perform best in most scenarios - even though for scenarios of easy or medium rewriting-complexity, and only as data grows larger, rewriting systems Rapid or ChaseGQR come in front. When looking across the mapping-language dimension, again Rulewerk seems to perform best in most of mapping categories due to its dominance on the small and medium-data sizes for rewriting-complexities of all difficulties. In fact, it also wins on scenarios of very-large data when the rewriting-complexity becomes difficult. Dominance of the chase seems to be inherent in this space; RDFox also wins for large data and difficult rewriting-complexities.

We performed the same analysis for just query-rewriting systems in Table 5(b) where the prepossessing times of ONTOP (and ChaseGQR - the only other rewriting system that supports preprocessing) were neglected. ONTOP here performs the best on hard rewriting-complexities, while Rapid is following close, especially on easy rewriting-complexities across all dimensions. ONTOP is outperformed in cases from other query rewriting systems and by nature cannot compete in the category of LAV mappings, reinforcing that query rewriting with LAV mappings is a future challenge.

Another notice in Table 5 is that ChaseGQR appears in multiple places, and although this is usually for easy complexity, 1-to-1 or LAV (which is what it was designed for), these results show that hybrid systems are promising for further investigation.

Comparing our results to literature, Rapid performs better end-to-end while ONTOP is faster when prepossessing is excluded, consistent to [56]. Moreover, differently to [64] in our scenarios Rapid is almost always faster or equal to Iqaros, agreeing with [56].

6.3 Key takeaways

Our experiments do not intend to exhibit a winner or a best system or approach. Some of the surprising results that we present intend to show the usefulness of our benchmarking framework and its ability to investigate assumptions in particular settings. Our reported results can be very well overthrown in different scenarios.

For the studied setting however, contrary to the commonly perceived opinion, the chase is a viable approach to query answering, and in many cases preferable to OBDA. When used over small to medium-sized datasets, the best-performing systems are Rulewerk and RDFox. As the data size grows, backward-chaining systems seem to come back in the game but only when the rewriting-complexity (i.e, query-ontology complexity) is on the easier side. This is probably because in the face of queries with small number of rewritings and larger data, a large number of tuples means the chase is more time-consuming to execute, but this eventually pays out against larger query rewritings, irrespective of the data size.

Rewriting algorithms performed better on the real-data scenario (NPD), which is a complex GAV scenario where usually the chase dominates. This probably due to the large arities of the sources indicating that chase systems should improve for large arities.

In terms of mapping-complexity, when considering one-to-one mappings where the complexity to the sources is negligible, forward-chaining systems seem costly compared to backward-chaining systems but again this changes as the rewriting-complexity raises. On the other hand, while forward-chaining systems perform best on complex GAV and LAV mappings on small to medium datasets, this is challenged when the dataset size increases. This might be because of the complex GQR and unfolding phases that the OBDA rewritings need to go through, and which might explode the final query size; or it might just be that chase is better with complex formulas. We hope these results are only the beginning of further investigations in these areas, such as looking into optimised LAV and GAV OBDA algorithms. Overall, T-Box rewriting dominates the 1-to-1 mapping scenarios, execution time is the bottleneck for GAV and view-based rewriting is slowing LAV down.

For chase systems, we have found that RDFox suffers more in the face of large numbers of labelled nulls. Nevertheless, the predominant benefit of chase systems is their ability to fully compute the materialisation of all entailed inferences in the data. Table 6 shows the effects of preprocessing, that is, if we ignore the loading and chase time for each run under the assumption that this only needs to be performed once. We can see that this provides a substantial performance increase in chase-based systems (last column of Table 6). In such situations where multiple queries must be answered (and the data remains static), completing the chase is advantageous.

Table 6: Impact of preprocessing time

	Average Time (ns)			
System	Total	Preprocessed	Δ	(%)
ChaseGQR	9.56E+09	9.41E+09	1.45E+08	(1.51%)
RDFox	1.25E+10	5.99E+08	1.19E+10	(95.22%)
Rulewerk	1.52E+10	9.48E+08	1.43E+10	(93.78%)
RST+ GQR	1.76E+11	1.73E+11	2.54E+09	(1.44%)
Ontop	2.10E+10	1.32E+10	7.79E+09	(37.17%)
RST+ Rulewerk	2.41E+10	7.33E+08	2.34E+10	(96.96%)

We have found that almost all scenarios from the query rewriting literature are actually fit for materialisation. ForBackBench also makes use of real-world data scenarios (such as NPD), which allow for the comparison of synthetic to real-world data, in order to guide the improvement of data generation for future use.

7 FUTURE WORK AND CONCLUSIONS

We presented the ForBackBench benchmarking framework, a cross-approach analysis environment that compares the forward- and backward-chaining approaches to query resolution, and evaluates the efficiency of commonly used OBDA and DE systems and scenarios. Looking at the evolution of these areas since the mid-1990, developing *good open-source tools for different components of data integration pipelines* is recognised as a first major challenge to move forward [29]. The ForBackBench framework provides multiple tools and converters to suit the required formats of various systems. ForBackBench currently supports a number of different languages and technologies including TGDs and queries in the chasebench or Rulewerk formats, OWL, RDF, OBDA-mappings SPARQL, and SQL and is built to be fully extensible, allowing the future addition and integration of newly developed systems and scenarios. Our work also necessitated the description of PJ-acyclic TGDs, a previously unknown, exact correspondence between LTGDs and DL-Lite$_R$.

Our findings suggest that chase-based approaches usually terminate successfully, and often perform faster than or on-par with OBDA approaches for smaller data, more complex mappings or worse rewriting-complexities. This leads us to recommend additional research into development of metrics that can differentiate between datasets that are more suitable for one approach above the other. We have clear directions for future work. Naturally, ForBackBench can be iteratively expanded to incorporate new scenarios and systems as they are developed, and our translation framework can be used to automate the work of cross-compatibility. Our ongoing plans include more elaborated analysis of different problem dimensions, e.g., query size or type of data used, supporting more general queries, e.g., with aggregation, incorporating hybrid approaches to query answering, e.g., [37], as well as more mapping languages such as R2RML [22].

ACKNOWLEDGMENTS

G. Konstantinidis work was partially supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1. T. Blount was partially supported by UoS's Web Science Insitute. A. Alhazmi was parially supported by the Fundamental Research Funds for Saudi Culture Mission in UK. We thank James Errington and Nathan Bharkhda for helping with early features of the implementation.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, Boston, US. http://webdam.inria.fr/Alice/

[2] Foto N. Afrati and Nikos Kiourtis. 2010. Computing certain answers in the presence of dependencies. *Information Systems* 35, 2 (2010), 149–169. https://doi.org/10.1016/j.is.2009.08.002 Special Section: Context-Oriented Information Integration.

[3] Rafi Ahmed, Philippe De Smedt, Weimin Du, William Kent, Mohammad A. Ketabchi, Witold A. Litwin, Abbas Rafii, and Ming Chien Shan. 1991. The Pegasus Heterogenous Multidatabase System. *IEEE Computer* 24, 12 (September 1991), 19–27.

[4] Roman Kontchakov Alessandro Artale, Diego Calvanese and Michael Zakharyaschev. 2009. The DL-Lite family and relations. *J. of Artificial Intelligence Research* 36 (2009), 1–69.

[5] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. 2008. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB Endowment* 1, 1 (2008), 230–244.

[6] Patricia C Arocena, Boris Glavic, Radu Ciucanu, and Renée J Miller. 2015. The iBench integration metadata generator. *Proceedings of the VLDB Endowment* 9, 3 (2015), 108–119.

[7] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. 2003. *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

[8] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *Rule Technologies: Foundations, Tools, and Applications*, Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman (Eds.). Springer International Publishing, Cham, 328–344.

[9] Timea Bagosi, Diego Calvanese, Josef Hardi, Sarah Komla-Ebri, Davide Lanti, Martin Rezk, Mariano Rodríguez-Muro, Mindaugas Slusnys, and Guohui Xiao. 2014. The Ontop Framework for Ontology Based Data Access. In *The Semantic Web and Web Science*, Dongyan Zhao, Jianfeng Du, Haofen Wang, Peng Wang, Donghong Ji, and Jeff Z. Pan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 67–77.

[10] Samantha Bail, Sandra Alkiviadous, Bijan Parsia, David Workman, Mark Van Harmelen, Rafael S Goncalves, and Cristina Garilao. 2012. FishMark: A linked data application benchmark. In *Joint Workshop on Scalable and High-Performance Semantic Web Systems, SSWS+ HPCSW 2012-In Conjunction with the International Semantic Web Conference, ISWC 2012*. RWTH Aachen University, ., 1–15.

[11] Pablo Barceló. 2009. Logical foundations of relational data exchange. *ACM SIGMOD Record* 38, 1 (2009), 49–58.

[12] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. Association for Computing Machinery, New York, NY, United States, 37–52.

[13] Christian Bizer and Andreas Schultz. 2009. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)* 5, 2 (2009), 1–24.

[14] Angela Bonifati, Ioana Ileana, and Michele Linardi. 2017. ChaseFUN: a Data Exchange Engine for Functional Dependencies at Scale. In *International Conference on Extending DatabaseTechnology (EDBT)*. OpenProceedings.org, Venice, Italy, 534–537. https://hal.archives-ouvertes.fr/hal-01979684

[15] Diego Calvanese, Jeremy Carroll, Giuseppe De Giacomo, Jim Hendler, Ivan Herman, Bijan Parsia, Peter Patel-Schneider, Alan Ruttenberg, Uli Sattler, Michael Schneider, and et al. 2012. OWL 2 Web Ontology Language Profiles (Second Edition). https://www.w3.org/TR/owl2-profiles/#OWL_2_QL

[16] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2006. Data Complexity of Query Answering in Description Logics. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning* (Lake District, UK) *(KR'06)*. AAAI Press, 260–270.

[17] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of automated reasoning* 39, 3 (2007), 385–429.

[18] David Carral, Irina Dragoste, Larry González, Ceriel Jacobs, Markus Krötzsch, and Jacopo Urbani. 2019. VLog: A Rule Engine for Knowledge Graphs. In *The Semantic Web – ISWC 2019*, Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtěch Svátek, Isabel Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon (Eds.). Springer International Publishing, Cham, 19–35.

[19] David Chaves-Fraga, Edna Ruckhaus, Freddy Priyatna, Maria-Esther Vidal, and Óscar Corcho. 2020. Enhancing OBDA Query Translation over Tabular Data with Morph-CSV. *CoRR* abs/2001.09052 (2020), arXiv–2001. https://arxiv.org/abs/2001.09052

[20] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. 2011. Optimized Query Rewriting for OWL 2 QL. In *Automated Deduction – CADE-23*, Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 192–206.

[21] Kendall Clark, Mike Grove, and Evren Sirin. 2021. *Stardog*. Stardog Union. http://stardog.com/

[22] Souripriya Das, Seema Sundara, and Richard Cyganiak. 2012. *R2RML: RDB to RDF mapping language*. W3C. http://www.w3.org/TR/r2rml/

[23] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration*. Elsevier.

[24] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. 2000. Recursive Query Plans for Data Integration. *Journal of Logic Programming* 43, 1 (2000), 49–73.

[25] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[26] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. 1999. Query optimization in the presence of limited access patterns. *ACM SIGMOD Record* 28, 2 (1999), 311–322.

[27] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. 1999. Navigational Plans For Data Integration. In *AAAI* (Orlando, Florida). 67–73.

[28] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment* 6, 9 (2013), 625–636.

[29] Behzad Golshan, Alon Halevy, George Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 101–106. https://doi.org/10.1145/3034786.3056124

[30] G. Gottlob, N. Leone, and F. Scarcello. 1999. Hypertree decompositions and tractable queries. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 21–32.

[31] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2011. Ontological Query Answering via Rewriting. In *Advances in Databases and Information Systems*, Johann Eder, Maria Bielikova, and A. Min Tjoa (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[32] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3, 2-3 (2005), 158–182.

[33] Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. 2012. Benchmarking Ontology-Based Query Rewriting Systems. *Proceedings of the AAAI Conference on Artificial Intelligence* 26, 1 (Jul. 2012). https://ojs.aaai.org/index.php/AAAI/article/view/8215

[34] David S Johnson and Anthony Klug. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and system Sciences* 28, 1 (1984), 167–189.

[35] George Konstantinidis and José Luis Ambite. 2011. 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, 97–108.

[36] George Konstantinidis and José Luis Ambite. 2014. Optimizing the chase: Scalable data integration under constraints. *Proceedings of the VLDB Endowment* 7, 14 (2014), 1869–1880.

[37] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. 2011. The combined approach to ontology-based data access. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*. AAAI Press, 2656–2661.

[38] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. 2015. The NPD benchmark: Reality check for OBDA systems. In *8th International Conference on Extending Database Technology (EDBT)*. OpenProceedings. org, Brussels, Belgium, 2135–2140.

[39] Matthias Lanzinger. 2021. Tractability Beyond ß-Acyclicity for Conjunctive Queries with Negation. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 355–369.

[40] Maurizio Lenzerini. 2002. Data Integration: A Theoretical Perspective. In *PODS*, Lucian Popa (Ed.). ACM, New York, NY, United States, 233–246.

[41] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey Ullman, and Murty Valiveti. 1998. Capability based mediation in TSIMMIS. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 564–566.

[42] Carsten Lutz, İnanç Seylan, David Toman, and Frank Wolter. 2013. The Combined Approach to OBDA: Taming Role Hierarchies Using Filters. In *The Semantic Web – ISWC 2013*, Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 314–330.

[43] Jose Mora and Oscar Corcho. 2013. Towards a Systematic Benchmarking of Ontology-Based Query Rewriting Systems. In *The Semantic Web – ISWC 2013*, Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 376–391.

[44] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. 2012. Usage-centric benchmarking of RDF triple stores. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press, Toronto, 2134–2140.

[45] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, et al. 2009. OWL 2 web ontology language profiles. *W3C recommendation* 27 (2009), 61.

[46] Jeffrey F Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D Ullman. 1989. Efficient evaluation of right-, left-, and multi-linear rules. *ACM SIGMOD Record* 18, 2 (1989), 235–242.

[47] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. 2015. RDFox: A Highly-Scalable RDF Store. In *The Semantic Web - ISWC 2015*, Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 3–20.

[48] University of Bozen-Bolzano. 2018. *OBDA File Structure*. Github. https://github.com/ontop/ontop/wiki/ObdaFileFormat

[49] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. 2009. Efficient Query Answering for OWL 2. In *The Semantic Web - ISWC 2009*, Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 489–504.

[50] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. 2009. A Comparison of Query Rewriting Techniques for DL-lite. *Description Logics* 477 (2009), 29.

[51] Héctor Pérez-Urbina, Edgar Rodríguez-Díaz, Michael Grove, George Konstantinidis, and Evren Sirin. 2012. Evaluation of Query Rewriting Approaches for OWL 2.. In *SSWS+ HPCSW@ ISWC*. Citeseer, Boston, USA, 32–44.

[52] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. 2014. TPC-DI: the first industry benchmark for data integration. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1367–1378.

[53] Rachel Pottinger and Alon Halevy. 2001. MiniCon:A Scalable Algorithm for Answering Queries Using Views. *The VLDB Journal* (2001).

[54] Carlos R. Rivero, Inma Hernández, David Ruiz, and Rafael Corchuelo. 2016. Mapping RDF knowledge bases using exchange samples. *Knowledge-Based Systems* 93 (2016), 47–66. https://doi.org/10.1016/j.knosys.2015.11.001

[55] Mariano Rodrıguez-Muro and Diego Calvanese. 2011. Dependencies: Making ontology based data access work in practice. *Proc. of AMW* 749 (2011).

[56] Mariano Rodríguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. 2013. Ontology-Based Data Access: Ontop of Databases. In *The Semantic Web – ISWC 2013*, Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 558–573.

[57] Riccardo Rosati and Alessandro Almatelli. 2010. Improving query answering over DL-Lite ontologies. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*. AAAI Press, Toronto, Canada, 290–300.

[58] Georgios Santipantakis, Konstantinos I. Kotis, and George A. Vouros. 2015. Accessing and Reasoning with Data from Disparate Data Sources Using Modular Ontologies and OBDA. In *Proceedings of the 11th International Conference on Semantic Systems* (Vienna, Austria) *(SEMANTICS '15)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/2814864.2814874

[59] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. 2014. OBDA: Query Rewriting or Materialization? In Practice, Both!. In *The Semantic Web – ISWC 2014*, Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble (Eds.). Springer International Publishing, Cham, 535–551.

[60] Gunjan Singh, Sumit Bhatia, and Raghava Mutharaju. 2020. OWL2Bench: A Benchmark for OWL 2 Reasoners. In *The Semantic Web – ISWC 2020*, Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal (Eds.). Springer International Publishing, Cham, 81–96.

[61] Jumah YJ Sleeman and Jehad Abdulhamid Hammad. 2020. A Review of Accessing Big Data with Significant Ontologies. *Knowledge Engineering and Data Science* 3, 2 (2020), 67–76.

[62] Jeffrey D. Ullman. 1997. Information Integration Using Logical Views. In *Proceedings of the Sixth International Conference on Database Theory*. Delphi, Greece, 19–40.

[63] Jacopo Urbani, Markus Krötzsch, Ceriel Jacobs, Irina Dragoste, and David Carral. 2018. Efficient Model Construction for Horn Logic with VLog. In *Automated Reasoning*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer International Publishing, Cham, 680–688.

[64] Tassos Venetis, Giorgos Stoilos, and Giorgos B Stamou. 2012. Incremental query rewriting for OWL 2 QL. In *Proceedings of the 2012 International Workshop on Description Logics*. Citeseer, Rome, Italy, 356–610.