Semantic Web Technologies: From Theory to Practice

Kumulative

HABILITATIONSSCHRIFT

zur Erlangung der Lehrbefugnis im Fach

INFORMATIONSSYSTEME

an der
Fakultät für Informatik
der
Technischen Universität Wien

vorgelegt von

Axel Polleres

Galway, Mai 2010

Contents

Dedicated to Inga & Aivi

Preface

"The truth is rarely pure and never simple." (Oscar Wilde) ... particularly on the Web.

The Semantic Web is about to grow up. Over the last few years technologies and standards to build up the architecture of this next generation of the Web have matured and are being deployed on large scale in many live Web sites. The underlying technology stack of the Semantic Web consists of several standards endorsed by the World Wide Web consortium (W3C) that provide the formal underpinings of a machine-readable "Web of Data" [94]:

- A Uniform Exchange Syntax: the eXtensible Markup Language (XML)
- A Uniform Data Exchange Format: the Resource Description Framework (RDF)
- Ontologies: RDF Schema and the Web Ontology Language (OWL)
- Rules: the Rule interchange format (RIF)
- Query and Transformation Languages: XQuery, SPARQL

The eXtensible Markup Language (XML)

Starting from the pure HTML Web which allowed mainly to exchange layout information for Web pages only, the introduction of the eXtensible Markup Language (XML) in its first edition in 1998 [19] meant a breakthrough for Web technologies. With XML as a uniform exchange syntax, any semi-structured data can be modeled as a tree. Along with available APIs, parsers and other tools, XML allows to define various other Web languages besides HTML. XML nowadays is not only the basis for Web data, but also for Web services [45] and is used in many custom applications as a convenient data exchange syntax. Schema description languages such as XML Schema [112] can be used to define XML languages; expressive query and transformation languages such as XQuery [27] and XSLT [68] allow to query specific parts of an XML tree, or to transform one XML language into another.

The Resource Description Framework (RDF)

The *Resource Description Framework* (RDF) – now around for over a decade already as well – is the basic data model for the Semantic Web. It is built upon one of the simplest structures for representing data: a directed labeled graph. An RDF graph is described by a set of triples of the form $\langle Subject\ Predicate\ Object \rangle$, also called *statements*, which represent the edges of this graph. Anonymous nodes in this graph – so called-blank nodes, akin to existential variables – allow to also model incomplete information. RDF's flat graph-like representation has the advantage of abstracting away from the data schema, and thus promises to allow for easier integration than customised XML data in different XML dialects: whereas the integration of different XML languages requires the transformation between different tree structures using transformation languages such as XSLT [68] or XQuery [27], different RDF graphs can simply be stored and queried alongside one another, and as soon as they share common nodes, form a joint graph upon a simple merge operation. While the normative syntax to exchange RDF, RDF/XML [13], is an XML dialect itself, there are various other serialisation formats for RDF, such as RDFa [1], a format that allows to embed RDF within (X)HTML, or non-XML representations such as the more readable Turtle [12] syntax; likewise RDF stores (e.g. YARS2 [54]) normally use their own, proprietary internal representations of triples, that do not relate to XML at all.

RDF Schema and the Web Ontology Language (OWL)

Although RDF itself is essentially schema-less, additional standards such as RDF Schema and OWL allow to formally describe the relations between the terms used in an RDF graph: i.e., the predicates in an RDF triple which form edges in an RDF graph (properties) and types of subject or object nodes in an RDF graph (classes). Formal descriptions of these properties and classes can be understood as logical theories, also called ontologies, which allow to infer new connections in an RDF graph, or link otherwise unconnected RDF graphs. Standard languages to describe ontologies on the Web are

- RDF Schema [20] – a lightweight ontology language that allows to describe essentially simple class hierarchies, as well as the domains and ranges of properties; and
- the Web Ontology language (OWL) [108] which was first published in 2004 and recently has been extended with additional useful features in the OWL2 [56] standard.

OWL offers richer means than RDF Schema to define formal relations between classes and properties, such as intersection and union of classes, value restrictions or cardinality restrictions. OWL2 offers even more features such as, for instance, the ability to define keys, property chains, or meta-modeling (i.e., speaking about classes as instances).

The Rule Interchange Format (RIF)

Although ontology languages such as OWL(2) offer a rich set of constructs to describe relations between RDF terms, these languages are still insufficient to express complex mappings between ontologies, which may better be described in terms of rule languages. The lack of standards in this area had been addressed by several proposals for rule languages on top of RDF, such as the Semantic Web Rule language (SWRL) [62], WRL [6], or N3 [14, 15]. These languages offer, for example, support for non-monotonic negation, or rich sets of built-in functions. The importance of rule languages – also outside the narrow use case of RDF rules – has finally lead to the establishment of another W3C working group in 2005 to standardise a generic Rule Interchange Format (RIF). RIF has recently reached proposed recommendation status and will soon be a W3C recommendation. The standard comprises several dialects such as (i) RIF Core [17], a minimal dialect close to Datalog, (ii) the RIF Basic Logic Dialect (RIF-BLD) [18] which offers the expressive features of Horn rules, and also (iii) a production rules dialect (RIF-PRD) [35]. A set of standard datatypes as well as built-in functions and predicates (RIF-DTB) are defined in a separate document [92]. The relation of RIF to OWL and RDF is detailed in another document [31] that defines the formal semantics of combinations of RIF rule sets with RDF graphs and OWL ontologies.

Query and Transformation Language: SPARQL

Finally, a crucial puzzle piece which pushed the recent wide uptake of Semantic Web technologies at large was the availability of a standard query language for RDF, namely SPARQL [97], which plays the same role for the Semantic Web as SQL does for relational data. SPARQL's syntax is roughly inspired by Turtle [12] and SQL [109], providing basic means to query RDF such as unions of conjunctive queries, value filtering, optional query parts, as well as slicing and sorting results. The recently re-chartered SPARQL1.1 W3C working group[1] aims at extending the original SPARQL language by commonly requested features such as aggregates, sub-queries, negation, and path expressions.

The work in the respective standardisation groups is partially still ongoing or only finished very recently. In parallel, there has been plenty of work in the scientific community to define the formal underpinnings for these standards:

- The logical foundations and properties of RDF and RDF Schema have been investigated in detail [83, 52, 89]. Correspondence of the formal semantics of RDF and RDF Schema [55] with Datalog and First-order logic have been studied in the literature [21, 22, 66].
- The semantics of standard fragments of OWL have been defined in terms of expressive Description Logics such as $\mathcal{SHOIN}$(D) (OWL DL) [61] or $\mathcal{SROIQ}$(D) (OWL2DL) [60], and the research on OWL has significantly influenced the Description Logics community over the past years: for example, in defining tractable fragments like the $\mathcal{EL}$ [8, 9] family of Description Logics, or fragments that allow to reduce basic reasoning tasks to query answering in SQL, such as the DL-Lite family of Description Logics [26]. Other fragments of OWL and OWL2 have been defined in terms of Horn rules such as DLP [51], OWL$^-$ [34], pD* [110], or Horn-SHIQ [72]. In fact, the new OWL2 specification defines tractable fragments of OWL based on these results: namely, OWL2EL, OWL2QL, and OWL2RL [79].
- The semantics of RIF builds on foundations such as Frame Logic [70] and Datalog. RIF borrows, e.g., notions of Datalog safety from the scientific literature to define fragments with finite minimal models despite the presence of built-ins: the *strongly-safe* fragment of RIF Core [17, Section 6.2] is inspired by a similar safety condition defined by Eiter, Schindlauer, et al. [39, 103]. In fact, the closely related area of decidable subsets of Datalog and answer set programs with function symbols is a very active field of research [10, 42, 25].
- The formal semantics of SPARQL is also very much inspired by academic results, such as by the seminal papers of Pérez et al. [85, 86]. Their work further lead to refined results on equivalences within SPARQL [104] and on the relation of SPARQL to Datalog [91, 90]. Angles and Gutierrez [7] later showed that SPARQL has exactly the expressive power of non-recursive safe Datalog with negation.

Likewise, the scientific community has identified and addressed gaps between the Semantic Web standards and the formal paradigms they are based on, which we want turn to next.

Gaps in the Semantic Web Architecture

Although the standards that make up the Semantic Web architecture have all been established by the W3C, they do not always integrate smoothly, indeed these standards had yet to prove useful

[1] `http://www.w3.org/2009/sparql/wiki`

"in the wild", i.e., to be applied on real Web data. Particularly, the following significant gaps have been identified in various works over the past years by the author and other researchers:

Gap 1: XML vs. RDF The jump from XML, which is a mere syntax format, to RDF, which is more declartive in nature, is not trivial, but needs to be addressed by appropriate – yet missing – transformation languages for exchanging information between RDF-based and XML-based applications.

Gap 2: RDF vs. OWL The clean conceptual model of Description Logics underlying the OWL semantics is not necessarily applicable directly to all RDF data, particularly to messy, potentially inconsistent data as found on the Web.

Gap 3: RDF/OWL vs. Rules/RIF There are several theoretical and practical concerns in combining ontologies and rules, such as decidability issues or how to merge classical open world reasoning with non-monotonic closed world inference. The current RIF specification leaves many of these questions open, subject to ongoing research.

Gap 4: SPARQL vs. RDF Schema/RIF/OWL Query answering over ontologies and rules and subtopics such as the semantics of SPARQL queries over RDF Schema and OWL ontologies, or querying over combinations of ontologies with RIF rulesets are still neglected by the current standards.

In the following, we will discuss these gaps in more depth, point out how they have been addressed in scientific works so far, and particularly how the work of the author has contributed.

Gap 1: XML vs. RDF

Although RDF's original normative syntax is an XML dialect, it proves impractical to view an RDF graph as an XML document: e.g., when trying to transform XML data in a custom format into RDF (lifting) or, respectively, RDF data into a specific XML schema (lowering) as may be required by a Web service: while W3C's SAWSDL [44] an GRDDL [29] working groups originally proposed XSLT for these tasks, the various ambiguous formats that RDF/XML can take to represent the same graph form an obstacle for defining uniform transformations [3]: to some extent, treating an RDF graph as an XML document contradicts the declarative nature of RDF. Several proposals to overcome the limitations in lifting and lowering by XSLT include (i) compiling SPARQL queries into XSLT [50], (ii) sequential applications of SPARQL and XSLT queries (via the intermediate step of SPARQL's result format [28], another XML format), or (iii) the extension of XSLT by special RDF access features [114] or SPARQL blocks [16]. The author of the present thesis and his co-authors have established another proposal: XSPARQL [3, 2], which is a new language integrating SPARQL and XQuery; this approach has the advantage of blending two languages that are conceptually very similar and facilitates more concise translations than the previous approaches. XSPARQL has recently been acknowledged as a member submission by the W3C [95, 71, 75, 84].

Gap 2: RDF vs. OWL

There is a certain "schism" between the core Semantic Web and Description Logics communities on what OWL shall be: the description of an ontology in RDF for RDF data, or an RDF exchange format for Description Logic theories. This schism manifests itself in the W3C's two orthogonal semantic specifications for OWL: OWL2's RDF-based semantics [105], which directly builds upon RDF's model-theoretic semantics [55], and OWL2's direct semantics [80], which builds upon the Description Logics $\mathcal{SROIQ}$ but is not defined for all RDF graphs. Both of them address different use cases; however, particular analyses on Web Data have shown [11, 58] that pure OWL(2) in its Description Logics based semantics is not practically applicable: (i) in published Web data we find a lot of non-DL ontologies [11], which only leave to apply the RDF-based semantics; (ii) data and ontologies found on the Web spread across different sources contain a lot of inconsistencies, which – in case one aims to still make sense out of this data – prohibits complete reasoning using Description Logics [58]; (iii) finally, current DL reasoners cannot deal with the amounts of instance data found on the Web, which is in the order of billions of statements. The approach included in the selected papers for the present thesis, SAOR (Scalable Authoritative OWL Reasoner) [59], aims at addressing these problems. SAOR provides incomplete, but arguably meaningful inferences over huge data sets crawled from the Web, based on rule-based OWL reasoning inspired by earlier approaches such as pD*[110], with further cautious modifications. Hogan and Decker [57] have later compared this approach to the new standard rule-based OWL2RL [79] profile, coming to the conclusion that OWL2RL, as a maximal fragment of OWL2 that can be formalised purely with Horn rules, runs into similar problems as Description Logics reasoning when taken as a basis for reasoning over Web data without the further modifications proposed in SAOR. An orthogonal approach to reason with real Web data [36] – also proposed by the author of this work together with Delbru, Tummarello and Decker – is likewise based on pD*, but applies inference in a modular fashion per dataset rather than over entire Web crawls.

Gap 3: RDF/OWL vs. Rules/RIF

Issues on combining RDF and/or OWL with rules, and particularly with rule sets expressed in RIF, have so far mostly been discussed on a theoretical level, perhaps because there has not yet been time enough for meaningful adoption of RIF on the Web.

One strand of these discussions is concerned with extending RDF with rules and constraints, in terms of either suggesting new non-standard rule languages for RDF to publish such rules [106, 15, 5, 6, 4], or theoretical considerations such as redundancy elimination with rules and constraints on top of RDF [78, 88]. An interesting side issue here concerns rule languages that allow existentials in the head such as RDFLog [23], or more recently Datalog$^{+/-}$ [24], which may in fact be viewed as a viable alternative or complement to purely Description Logics based ontology languages. Non-monotonicity – which is not considered in OWL, but is available in most of the suggested rule languages for RDF [5, 15, 6] by incorporating a form of "negation as failure" – has sparked a lot of discussions in the Semantic Web community, since it was viewed as inadequate for an open environment such as the Web by some, whereas others (including the author of the present work) argued that "scoped negation" [69, 93] – that is, non-monotonic negation applied over a fixed, scoped part of the Web – was very useful for many Web data applications. This is closely related to what Etzioni et al. [43] called the "local closed world assumption" in earlier work.

Another quite significant strand of research has developed on the theoretical combination of Description Logics and (non-monotonic) rules in a joint logical framework. While the naïve combination of even Horn rules without function symbols and ontologies in quite inexpressive Description Logics loses the desirable decidability properties of the latter [74], there have been several proposals for decidable fragments of this combination [51, 82, 72] or even extending the idea of such decidable combinations to rules with non-monotonic negation [98, 99, 101, 81, 77]. Another decidable approach was to define the semantic interplay between ontologies and rules via a narrow, query-like interface within rule bodies [40]. Aside from considerations about decidability, there have been several proposals for what would be the right logical framework to embed combinations of classical logical theories (which DL ontologies fall into) and non-monotonic rule languages. These include approaches based on MKNF [81], FO-AEL [32], or Quantified Equilibrium Logics (QEL) [33], the latter of which is included in the collection of papers selected for the present thesis. For an overview of issues concerned with combining ontologies and rules, we also refer to surveys of existing approaches in [38, 37, 100], some of which the author contributed to.

As a side note, it should be mentioned that rule-based/resolution-based reasoning has been very successfully applied in implementing Description Logics or OWL reasoners in approaches such as KAON2 [63] and DLog [76] which significantly outperform tableaux-based DL reasoners on certain problems (particularly instance reasoning).

Gap 4: SPARQL vs. RDF Schema/RIF/OWL

SPARQL has in its official specification only been defined as a query language over RDF graphs, not taking into account RDF Schema, OWL ontologies or RIF rule sets. Although the official specification defines frame conditions for extending SPARQL by higher entailment regimes [97, Section 12.6], few works have actually instantiated this mechanism and defined how SPARQL should handle ontologies and rule sets.

As for OWL, conjunctive query answering over expressive description logics is a topic of active research in the Description Logics Community, with important insights only being very recent [41, 47, 46, 73], none of which yet having covered the Description Logics underlying OWL and OWL2, i.e. $\mathcal{SHOIN}$(D) and $\mathcal{SROIQ}$(D). Answering full SPARQL queries on top of OWL has only preliminarily been addressed in the scientific community [107, 67] so far.

In terms of SPARQL on top of RDF in combination with rule sets, the choices are more obvious. Firstly, as mentioned above, SPARQL itself can be translated to non-recursive rules – more precisely into non-recursive Datalog with negation [91, 7]. Secondly, expanding on the translation from [91], additional RDF rule sets that guarantee a finite closure, such as Datalog style rules on top of RDF, can be allowed, covering a significant subset of RIF or rule-based approximations of RDFS and OWL [65, 64]. Two of the works dealing with these matters [91, 64] are included in the selected papers of this thesis.

One should mention here that certain SPARQL queries themselves may be read as rules: that is, SPARQL's CONSTRUCT queries facilitate the generation of new RDF triples (defined in a CONSTRUCT template that plays the role of the rule head), based on the answers to a graph pattern (that plays the role of a rule body). This idea has been the basis for proposals to extend RDF to so-called Networked Graphs [102] or Extended RDF graphs [96], that enable the inclusion of implicit knowledge defined as SPARQL CONSTRUCT queries. Extending RDF graphs in such fashions has also been proposed as an expressive means to define ontology mappings by the author of this thesis [96], where the respective contribution is also included in the selected papers.

The recently started W3C SPARQL1.1 working group, co-chaired by the author of the present thesis, has published a working draft summarising first results on defining an OWL entailment regime for SPARQL [49], which, although worth to be mentioned, will not necessarily encompass full conjunctive queries with non-distinguished variables.

Selected Papers

The present habilitation thesis comprises a collection of articles reflecting the author's contribution in addressing a number of relevant research problems to close the above mentioned gaps in the Semantic Web architecture.

The first paper, "*A Semantical Framework for Hybrid Knowledge Bases*" [33], co-authored with Jos de Bruijn, David Pearce and Agustín Valverde contributes to the fundamental discussion of a logical framework for combining classical theories (such as DL ontologies) with logic programs involving non-monotonic negation, in this case under the (open) answer set semantics. Based on initial discussions among the author and David Pearce, the founder of Equilibrium Logics (a non-classical logic which can be viewed as the base logic of answer set programming), we came to the conclusion that Quantified Equilibrium Logics (QEL), a first-order variant of EL, is a promising candidate for the unifying logical framework in quest. In the framework of QEL, one can either enforce or relax the unique names assumption, or – by adding axiomatisations that enforce the law of the excluded middle for certain predicates – make particular predicates behave in a "classical" manner whereas others are treated non-monotonically in the spirit of (open) answer set programming. We showed that the defined embedding of hybrid knowledge bases in the logical framework of QEL encompasses previously defined operational semantics by Rosati [98, 99, 101]. This correspondence naturally provides decidable fragments of QEL. At the same time, concepts such as strong equivalence, which are well-investigated in the framework of answer set programming, carry over to hybrid knowledge bases embedded in the framework of QEL. This work particularly addresses theoretical aspects of *Gap 3: RDF/OWL vs. Rules/RIF*.

Another line of research addressing *Gap 4: SPARQL vs. RDF Schema/RIF/OWL* is presented in the following three works.

The second paper, "*From SPARQL to Rules (and back)*" [91] clarifies the relationship of SPARQL to Datalog. Besides providing a translation from SPARQL to non-recursive Datalog with negation, several alternative join semantics for SPARQL are discussed, which – at the time of publication of this paper and prior to SPARQL becoming a W3C recommendation – were not yet entirely fixed. Additionally, the paper sketches several useful extensions of SPARQL by adding rules or defining some extra operators such as MINUS.

The third paper, "*SPARQL++ for Mapping between RDF Vocabularies*" [96], co-authored with Roman Schindlauer and François Scharffe continues this line of research. Based on the idea of reductions to answer set programming as a superset of Datalog, we elaborate on several SPARQL extensions such as aggregate functions, value construction, or what we call Extended Graphs: i.e., RDF graphs that include implicit knowledge in the form of SPARQL queries which are interpreted as "views". We demonstrate that the proposed extensions can be used to model Ontology mappings not expressible in OWL. It is worthwhile to mention that at least the first two new features (aggregates and value construction) – which were not present in SPARQL's original specification but easy to add in our answer set programming based framework – are very likely to be added in a similar form to SPARQL1.1.[2]

The fourth paper, "*Dynamic Querying of Mass-Storage RDF Data with Rule-Based Entailment Regimes*"[64], co-authored with Giovambattista Ianni, Thomas Krennwallner, and Alessandra Martello, expands on the results of the second paper in a different direction, towards providing an efficient implementation of the approach. In particular, we deploy a combination of the DLV-DB system [111] and DLVHEX [39], and exploit magic sets optimisations inherent in DLV to improve on the basic translation from [91] on RDF data stored in a persistent repository. Moreover, the paper defines a generic extension of SPARQL by rule-based entailment regimes, which the implemented system allows to load dynamically for each SPARQL query: the system allows users to query data dynamically with different ontologies under different (rule-based) entailment regimes. To the best of our knowledge, most existing RDF Stores only provide fixed pre-materialised inference, whereas we could show in this paper that – by thorough design – dynamic inferencing can still be relatively efficient.

The fifth paper, "*Scalable Authoritative OWL Reasoning for the Web*" [59] co-authored with Aidan Hogan and Andreas Harth, addresses *Gap 2: RDF vs. OWL* by defining practically viable OWL inference on Web data: similar to the previous paper, this work goes also in the direction of implementing efficient inference support, this time though following a pre-materialisation approach by forward-chaining. The reason for this approach is a different use case than before: the goal here is to provide indexed pre-computed inferences on an extremely large dataset in the order of billions of RDF triples to be used in search results for the Semantic Web Search Engine (SWSE) project. The paper defines a special RDF rule set that is inspired by ter Horst's pD*, but tailored for reasoning per sorting and file scans, i.e., (i) by extracting the ontological part (T-Box) of the dataset which is relatively small and can be kept in memory, and (ii) avoiding expensive joins on the instance data (A-Box) where possible. We conclude that all of the RDFS rules and most of the OWL rules fall under a category of rules that does not require A-Box joins. As a second optimisation, the application of rules is triggered only if the rule is authoritatively applicable, avoiding a phenomenon which we call "ontology hijacking": i.e., uncontrolled redefinition of ontologies by third parties that can lead to potentially harmful inferences. In order to achieve this, we introduce the notion of a so-called T-Box split-rule – a rule which has a body divided into an A-Box and T-Box part – along with an intuitive definition of authoritative rule application for split rules. Similar approaches to do scalable reasoning on large sets of RDF data have been independently presented since, demonstrating that our core approach can be naturally applied in a distributed fashion [115, 113]. None of these other approaches go beyond RDFS reasoning and neither apply the concept of authoritativeness, which proves very helpful on the Web to filter out bogus inferences from noisy Web data; in fact, both approaches [115, 113] only have been evaluated on synthetic data.

Finally, the sixth paper "*XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage*"[3], co-authored by Waseem Akhtar, Jacek Kopecký and Thomas Krennwallner, intends to close *Gap 1: XML vs. RDF* by defining a novel query language which is the merge of XQuery and SPARQL. We demonstrate that XSPARQL provides concise and intuitive solutions for mapping between XML and RDF in either direction. The paper also describes

[2]cf. `http://www.w3.org/2009/05/sparql-phase-II-charter.html`, here value construction is subsumed under the term "project expressions".

an initial implementation of an XSPARQL engine, available for user evaluation.[3] This paper had been among the nominees for the best paper award at the 5th European Semantic Web Conference (ESWC2008). The approach has experienced considerable attention and been extended later on to a W3C member submission under the direction of the thesis' author [95, 71, 75, 84]. In the paper included here, we also include the formal semantics of XSPARQL, which was not published in [3] originally, but as a part of the W3C member submission [71].

Additionally, the author has conducted work on foundations as well as practical applications of Semantic Web technologies that is not contained in this collection, some of which is worthwhile to be mentioned in order to emphasise the breath of the author's work in this challenging application field for information systems.

In [93], different semantics for rules with "scoped negation" were proposed, based on reductions to the stable model semantics and the well-founded semantics. The author conducted the major part of this work, based on discussions with two students, Andreas Harth and Cristina Feier.

In [87], we have presented a conceptual model and implementation for a simple workflow language on top of SPARQL, along with a visual editor, which allows to compose reusable data-mashups for RDF data published on the Web. This work was a collaboration with several researchers at the National University of Ireland, Galway (Danh Le Phuoc, Manfred Hauswirth, Giovanni Tummarello) where the author's contribution was in defining the underlying conceptual model and base operators of the presented workflow language.

In [30], we have developed extensions to the widely used open-source content management system Drupal,[4] in order to promote the uptake of Semantic Web technologies. This work has won the best paper award in the in-use track of last year's International Semantic Web Conferece (ISWC2009). Furthermore, the effort has eventually lead to the integration of RDF technologies into Drupal 7 Core, potentially affecting over 200,000 Web sites currently using Drupal 6. The work was conducted in collaboration with Stéphane Corlosquet, a master student under the author's supervision, and based on input from and discussions with Stefan Decker, Renaud Delbru and Tim Clark, where the author provided the core mapping of Drupal's content model to OWL, co-developed a model to import external data from external SPARQL endpoints and provided overall direction of the project.

In [53], the author together with colleagues from the National University of Ireland, Galway (Jürgen Umbrich, Marcel Karnstedt), the University of Ilmenau (Kai-Uwe Sattler), University of Karlsruhe (Andreas Harth), and the Max-Planck Institute in Saarbrücken (Katja Hose) have presented a novel approach to perform live queries on RDF Data published across multiple sources across the Web. In order to define a reasonable middle-ground between storing crawled data in a large centralised index – like most current search engines perform – or directly looking up known sources on demand, we propose to use lightweight data summaries based on QTrees for source selection, with promising initial results. The author's contribution in this work was in providing conceptual guidance in terms of the different characteristics of linked RDF data with common database scenarios where QTrees have been applied earlier, whereafter the ideas of the paper were developed jointly in a very interactive fashion among all contributors.

[3]cf. `http://xsparql.deri.org/`

[4]http://drupal.org/

Acknowledgements

I would like to express my deepest gratitude to all people who helped me get this far, beginning with the co-supervisors of my diploma and doctoral theses, Nicola Leone, Wolfgang Faber, and finally Thomas Eiter, who also encouraged and mentored me in the process of submitting the present habilitation thesis.

I further have to thank Dieter Fensel for initially bringing me in touch with the exciting world of the Semantic Web, and all my work colleagues over the past years – many of which became close friends – from Vienna University of Technology, University of Innsbruck, and Universidad Rey Juan Carlos, who made research over all these years so pleasant and interesting in hours of discussions about research topics and beyond. Of course, I also want to especially thank my present colleagues in the Digital Enterprise Research Institute (DERI) at the National University of Ireland, Galway, foremost Stefan Decker and Manfred Hauswirth, whose enthusiasm and vision were most inspiring over the last three years since working in DERI.

I want to thank all my co-authors, especially those involved in the articles which were selected for the present thesis, namely, Jos, Andreas, Thomas, GB, Alessandra, Aidan, David, Agustín, Roman, François, Waseem, and Jacek, as well as my students Nuno, Jürgen, Stéphane, Lin, Philipp, and finally Antoine Zimmerman, who works with me as a postdoctoral researcher.

Last, but not least, I want to thank my parents Mechthild and Herbert and my sister Julia for all their love and support over the years, and finally my wife Inga and my little daughter Aivi for making every day worth it all.

The work presented in this thesis has been supported in parts by (i) Science Foundation Ireland – under the Líon (SFI/02/CE1/I131) and Líon-2 (SFI/08/CE/I1380) projects, (ii) by the Spanish Ministry of Education – under the projects projects TIC-2003-9001, URJC-CM-2006-CET-0300, as well as a "Juan de la Cierva" postdoctoral fellowship, and (iii) by the EU under the FP6 projects Knowledge Web (IST-2004-507482) and inContext (IST-034718).

References

[1] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. W3C recommendation, W3C, October 2008. Available at `http://www.w3.org/TR/rdfa-syntax/`.

[2] Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and **Axel Polleres**. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. Technical Report DERI-TR-2007-12-14, DERI Galway, 2007. Available at `http://www.deri.ie/fileadmin/documents/TRs/DERI-TR-2007-12-14.pdf`.

[3] Waseem Akhtar, Jacek Kopecky, Thomas Krennwallner, and **Axel Polleres**. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. In *Proceedings of the 5th European Semantic Web Conference (ESWC2008)*, pages 432–447, Tenerife, Spain, June 2008. Springer.

[4] Anastasia Analyti, Grigoris Antoniou, and Carlos Viegas Damásio. A principled framework for modular web rule bases and its semantics. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 390–400, 2008.

[5] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner. Extended RDF as a semantic foundation of rule markup languages. *Journal of Artificial Intelligence Research*, 32:37–94, 2008.

[6] Jürgen Angele, Harold Boley, Jos de Bruijn, Dieter Fensel, Pascal Hitzler, Michael Kifer, Reto Krummenacher, Holger Lausen, **Axel Polleres**, and Rudi Studer. Web Rule Language (WRL), September 2005. W3C member submission.

[7] Renzo Angles and Claudio Gutierrez. The expressive power of sparql. In *International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129, Karlsruhe, Germany, 2008. Springer.

[8] Franz Baader. Terminological cycles in a description logic with existential restrictions. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI2003)*, pages 325–330, Acapulco, Mexico, August 2003.

[9] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI2005)*, pages 364–369, Edinburgh, Scotland, UK, July 2005. Professional Book Center.

[10] Sabrina Baselice, Piero A. Bonatti, and Giovanni Criscuolo. On finitely recursive programs. *TPLP*, 9(2):213–238, 2009.

[11] Sean Bechhofer and Raphael Volz. Patching syntax in OWL ontologies. In *International Semantic Web Conference (ISWC 2004)*, pages 668–682, Hiroshima, Japan, November 2004.

[12] Dave Beckett and Tim Berners-Lee. Turtle – Terse RDF Triple Language. W3C team submission, W3C, January 2008. Available at `http://www.w3.org/TeamSubmission/turtle/`.

[13] Dave Beckett and Brian McBride. RDF/XML Syntax Specification (Revised). W3C recommendation, W3C, February 2004. Available at `http://www.w3.org/TR/REC-rdf-syntax/`.

[14] Tim Berners-Lee and Dan Connolly. Notation3 (N3): A readable RDF syntax. W3C team submission, W3C, January 2008. Available at `http://www.w3.org/TeamSubmission/n3/`.

[15] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3logic: A logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8(3):249–269, 2008.

[16] Diego Berrueta, Jose E. Labra, and Ivan Herman. XSLT+SPARQL : Scripting the Semantic Web with SPARQL embedded into XSLT stylesheets. In Chris Bizer, Sören Auer, Gunnar Aastrand Grimmes, and Tom Heath, editors, *4th Workshop on Scripting for the Semantic Web*, Tenerife, June 2008.

[17] Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, **Axel Polleres**, and Dave Reynolds. RIF Core Dialect. W3C proposed recommendation, W3C, May 2010. Available at `http://www.w3.org/TR/2010/PR-rif-core-20100511/`.

[18] Harold Boley and Michael Kifer. RIF Basic Logic Dialect. W3C proposed recommendation, W3C, May 2010. Available at `http://www.w3.org/TR/2010/PR-rif-bld-20100511/`.

[19] Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, W3C, February 1998. Available at `http://www.w3.org/TR/1998/REC-xml-19980210`.

[20] Dan Brickley, R. Guha, and Brian McBride (eds.). RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C, February 2004. W3C Recommendation.

[21] Jos de Bruijn, Enrico Franconi, and Sergio Tessaris. Logical reconstruction of normative RDF. In *OWL: Experiences and Directions Workshop (OWLED-2005)*, Galway, Ireland, November 2005.

[22] Jos de Bruijn and Stijn Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, number 4825 in Lecture Notes in Computer Science, pages 86–99, Busan, Korea, November 2007. Springer.

[23] François Bry, Tim Furche, Clemens Ley, Benedikt Linse, and Bruno Marnette. RDFLog: It's like datalog for RDF. In *Proceedings of 22nd Workshop on (Constraint) Logic Programming, Dresden (30th September–1st October 2008)*, 2008.

[24] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Tractable query answering over ontologies with datalog$^{+/-}$. In *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*, Oxford, UK, July 2009.

[25] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Magic sets for the bottom-up evaluation of finitely recursive programs. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference (LPNMR 2009)*, volume 5753 of *Lecture Notes in Computer Science*, pages 71–86, Potsdam, Germany, September 2009. Springer.

[26] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[27] Don Chamberlin, Jonathan Robie, Scott Boag, Mary F. Fernández, Jérôme Siméon, and Daniela Florescu. XQuery 1.0: An XML Query Language. W3C recommendation, W3C, January 2007. W3C Recommendation, available at `http://www.w3.org/TR/xquery/`.

[28] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. SPARQL Protocol for RDF. W3C recommendation, W3C, January 2008. Available at `http://www.w3.org/TR/rdf-sparql-protocol/`.

[29] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation, W3C, September 2007. Available at `http://www.w3.org/TR/sawsdl/`.

[30] Stéphane Corlosquet, Renaud Delbru, Tim Clark, **Axel Polleres**, and Stefan Decker. Produce and consume linked data with drupal! In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*, volume 5823 of *Lecture Notes in Computer Science*, pages 763–778, Washington DC, USA, October 2009. Springer.

[31] Jos de Bruijn. RIF RDF and OWL Compatibility. W3C proposed recommendation, W3C, May 2010. Available at `http://www.w3.org/TR/2010/PR-rif-rdf-owl-20100511/`.

[32] Jos de Bruijn, Thomas Eiter, **Axel Polleres**, and Hans Tompits. Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 304–309, Hyderabad, India, January 2007. AAAI.

[33] Jos de Bruijn, David Pearce, **Axel Polleres**, and Agustín Valverde. A semantical framework for hybrid knowledge bases. *Knowledge and Information Systems*, Special Issue: RR 2007, 2010. Accepted for publication.

[34] Jos de Bruijn, **Axel Polleres**, Rubén Lara, and Dieter Fensel. OWL$^-$. Final draft d20.1v0.2, WSML, 2005.

[35] Christian de Sainte Marie, Gary Hallmark, and Adrian Paschke. RIF Production Rule Dialect. W3C proposed recommendation, W3C, May 2010. Available at `http://www.w3.org/TR/2010/PR-rif-prd-20100511/`.

[36] Renaud Delbru, **Axel Polleres**, Giovanni Tummarello, and Stefan Decker. Context dependent reasoning for semantic documents in sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, Karlsruhe, Germany, October 2008.

[37] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and **Axel Polleres**. Rules and ontologies for the semantic web. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web 2008*, volume 5224 of *Lecture Notes in Computer Science*, pages 1–53. Springer, San Servolo Island, Venice, Italy, September 2008.

[38] Thomas Eiter, Giovambattista Ianni, **Axel Polleres**, Roman Schindlauer, and Hans Tompits. Reasoning with rules and ontologies. In P. Barahona et al., editor, *Reasoning Web 2006*, volume 4126 of *Lecture Notes in Computer Science*, pages 93–127. Springer, September 2006.

[39] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *LNCS*, pages 273–287, Budva, Montenegro, June 2006. Springer.

[40] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, Whistler, Canada, 2004. AAAI Press.

[41] Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in description logics with transitive roles. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 759–764, Pasadena, California, USA, July 2009.

[42] Thomas Eiter and Mantas Simkus. Fdnc: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Trans. Comput. Log.*, 11(2), 2010.

[43] Oren Etzioni, Keith Golden, and Daniel Weld. Tractable closed world reasoning with updates. In *KR'94: Principles of Knowledge Representation and Reasoning*, pages 178–189, San Francisco, California, 1994. Morgan Kaufmann.

[44] Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C, August 2007. Available at `http://www.w3.org/TR/sawsdl/`.

[45] Dieter Fensel, Holger Lausen, **Axel Polleres**, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer, 2006.

[46] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in SHOQ. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008*, pages 252–262, Sydney, Australia, September 2008. AAAI Press.

[47] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic SHIQ. *J. Artif. Intell. Res. (JAIR)*, 31:157–204, 2008.

[48] Birte Glimm and Sebastian Rudolph. Status QIO: Conjunctive Query Entailment is Decidable. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010*, pages 225–235, Toronto, Canada, May 2010. AAAI Press.

[49] Birte Glimm, Chimezie Ogbuji, Sandro Hawke, Ivan Herman, Bijan Parsia, Axel Polleres, and Andy Seaborne. SPARQL 1.1 Entailment Regimes. W3C working draft, W3C, May 2010. Available at `http://www.w3.org/TR/sparql11-entailment/`.

[50] Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke. Embedding SPARQL into XQuery/XSLT. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, pages 2271–2278, Fortaleza, Ceara, Brazil, March 2008. ACM.

[51] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *12th International Conference on World Wide Web (WWW'03)*, pages 48–57, Budapest, Hungary, 2003. ACM.

[52] Claudio Gutiérrez, Carlos A. Hurtado, and Alberto O. Mendelzon. Foundations of Semantic Web Databases. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004)*, pages 95–106, Paris, France, 2004. ACM.

[53] Andreas Harth, Katja Hose, Marcel Karnstedt, **Axel Polleres**, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th World Wide Web Conference (WWW2010)*, Raleigh, NC, USA, April 2010. ACM Press. Technical report version available at `http://www.deri.ie/fileadmin/documents/DERI-TR-2009-11-17.pdf`.

[54] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A federated repository for querying graph structured data from the web. In *6th International Semantic Web Conference, 2nd Asian Semantic Web Conference*, pages 211–224, 2007.

[55] Patrick Hayes. RDF semantics. Technical report, W3C, February 2004. W3C Recommendation.

[56] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer. W3C recommendation, W3C, October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[57] Aidan Hogan and Stefan Decker. On the ostensibly silent 'W' in OWL 2 RL. In *Web Reasoning and Rule Systems – Third International Conference, RR 2009*, pages 118–134, 2009.

[58] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In *3rd International Workshop on Linked Data on the Web (LDOW2010) at WWW2010*, Raleigh, USA, April 2010.

[59] Aidan Hogan, Andreas Harth, and **Axel Polleres**. Scalable authoritative OWL reasoning for the Web. *International Journal on Semantic Web and Information Systems*, 5(2):49–90, 2009.

[60] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 57–67. AAAI Press, 2006.

[61] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[62] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2004. W3C member submission.

[63] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing shiq-description logic to disjunctive datalog programs. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pages 152–162, Whistler, Canada, 2004. AAAI Press.

[64] Giovambattista Ianni, Thomas Krennwallner, Alessandra Martello, and Axel Polleres. Dynamic querying of mass-storage RDF data with rule-based entailment regimes. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference (ISWC 2009)*, volume 5823 of *Lecture Notes in Computer Science*, pages 310–327, Washington DC, USA, October 2009. Springer.

[65] Giovambattista Ianni, Thomas Krennwallner, Alessandra Martello, and Axel Polleres. A rule system for querying persistent RDFS data. In *Proceedings of the 6th European Semantic Web Conference (ESWC2009)*, Heraklion, Greece, May 2009. Springer. Demo Paper.

[66] Giovambattista Ianni, Alessandra Martello, Claudio Panetta, and Giorgio Terracina. Efficiently querying RDF(S) ontologies with Answer Set Programming. *Journal of Logic and Computation (Special issue)*, 19(4):671–695, August 2009.

[67] Yixin Jing, Dongwon Jeong, and Doo-Kwon Baik. SPARQL graph pattern rewriting for OWL-DL inference queries. *Knowl. Inf. Syst.*, 20(2):243–262, 2009.

[68] Michael Kay. XSL Transformations (XSLT) Version 2.0 . W3C Recommendation, W3C, January 2007. Available at `http://www.w3.org/TR/xslt`20.

[69] Michael Kifer. Nonmonotonic reasoning in FLORA-2. In *8th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, Diamante, Italy, 2005. Invited Paper.

[70] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[71] Thomas Krennwallner, Nuno Lopes, and **Axel Polleres**. XSPARQL: Semantics, January 2009. W3C member submission.

[72] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Complexity boundaries for horn description logics. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pages 452–457, Vancouver, British Columbia, Canada, July 2007.

[73] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, pages 310–323, Busan, Korea, November 2007.

[74] Alon Y. Levy and Marie-Christine Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165–209, 1998.

[75] Nuno Lopes, Thomas Krennwallner, **Axel Polleres**, Waseem Akhtar, and Stéphane Corlosquet. XSPARQL: Implementation and Test-cases, January 2009. W3C member submission.

[76] Gergely Lukácsy and Péter Szeredi. Efficient description logic reasoning in Prolog: the DLog system. *Theory and Practice of Logic Programming*, 9(3):343–414, 2009.

[77] Thomas Lukasiewicz. A novel combination of answer set programming with description logics for the semantic web. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2010. In press.

[78] Michael Meier. Towards Rule-Based Minimization of RDF Graphs under Constraints. In *Proc. RR'08*, volume 5341 of *LNCS*, pages 89–103. Springer, 2008.

[79] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Casrsten Lutz, Diego Calvanese, Jeremy Carroll, Guiseppe De Giacomo, Jim Hendler, Ivan Herman, Bijan Parsia, Peter F. Patel-Schneider, Alan Ruttenberg, Uli Sattler, and Michael Schneider. OWL 2 Web Ontology Language Profiles. W3C recommendation, W3C, October 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.

[80] Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Uli Sattler. OWL 2 Web Ontology Language Direct Semantics. W3C recommendation, W3C, October 2009. Available at `http://www.w3.org/TR/owl2-direct-semantics/`.

[81] Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 477–482, Hyderabad, India, January 6–12 2007. AAAI.

[82] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[83] Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal deductive systems for RDF. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 53–67, Innsbruck, Austria, June 2007. Springer.

[84] Alexandre Passant, Jacek Kopecký, Stéphane Corlosquet, Diego Berrueta, Davide Palmisano, and **Axel Polleres**. XSPARQL: Use cases, January 2009. W3C member submission.

[85] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International Semantic Web Conference (ISWC 2006)*, pages 30–43, 2006.

[86] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems*, 34(3):Article 16 (45 pages), 2009.

[87] Danh Le Phuoc, **Axel Polleres**, Giovanni Tummarello, Christian Morbidoni, and Manfred Hauswirth. Rapid semantic web mashup development through semantic web pipes. In *Proceedings of the 18th World Wide Web Conference (WWW2009)*, pages 581–590, Madrid, Spain, April 2009. ACM Press.

[88] Reinhard Pichler, **Axel Polleres**, Sebastian Skritek, and Stefan Woltran. Minimising RDF graphs under rules and constraints revisited. In *4th Alberto Mendelzon Workshop on Foundations of Data Management*, May 2010. To appear, technical report version available at `http://www.deri.ie/fileadmin/documents/DERI-TR-2010-04-23.pdf`.

[89] Reinhard Pichler, **Axel Polleres**, Fang Wei, and Stefan Woltran. Entailment for domain-restricted RDF. In *Proceedings of the 5th European Semantic Web Conference (ESWC2008)*, pages 200–214, Tenerife, Spain, June 2008. Springer.

[90] **Axel Polleres**. SPARQL Rules! Technical Report GIA-TR-2006-11-28, Universidad Rey Juan Carlos, Móstoles, Spain, 2006. Available at `http://www.polleres.net/TRs/GIA-TR-2006-11-28.pdf`.

[91] **Axel Polleres**. From SPARQL to rules (and back). In *Proceedings of the 16th World Wide Web Conference (WWW2007)*, pages 787–796, Banff, Canada, May 2007. ACM Press. Extended technical report version available at `http://www.polleres.net/TRs/GIA-TR-2006-11-28.pdf`, slides available at `http://www.polleres.net/publications/poll-2007www-slides.pdf`.

[92] **Axel Polleres**, Harold Boley, and Michael Kifer. RIF Datatypes and Built-Ins 1.0. W3C proposed recommendation, W3C, May 2010. Available at `http://www.w3.org/TR/2010/PR-rif-dtb-20100511/`.

[93] **Axel Polleres**, Cristina Feier, and Andreas Harth. Rules with contextually scoped negation. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.

[94] **Axel Polleres** and David Huynh, editors. *Journal of Web Semantics, Special Issue: The Web of Data*, volume 7(3). Elsevier, 2009.

[95] **Axel Polleres**, Thomas Krennwallner, Nuno Lopes, Jacek Kopecký, and Stefan Decker. XSPARQL Language Specification, January 2009. W3C member submission.

[96] **Axel Polleres**, François Scharffe, and Roman Schindlauer. SPARQL++ for mapping between RDF vocabularies. In *OTM 2007, Part I : Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 878–896, Vilamoura, Algarve, Portugal, November 2007. Springer.

[97] Eric Prud′hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation, W3C, January 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

[98] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1):61–73, 2005.

[99] Riccardo Rosati. Semantic and computational advantages of the safe integration of ontologies and rules. In *Proceedings of the Third International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, volume 3703 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005.

[100] Riccardo Rosati. Integrating Ontologies and Rules: Semantic and Computational Issues. In Pedro Barahona, François Bry, Enrico Franconi, Ulrike Sattler, and Nicola Henze, editors, *Reasoning Web, Second International Summer School 2006, Lissabon, Portugal, September 25-29, 2006, Tutorial Lectures*, volume 4126 of *LNCS*, pages 128–151. Springer, September 2006.

[101] Riccardo Rosati. $\mathcal{DL} + log$: Tight integration of description logics and disjunctive datalog. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 68–78, 2006.

[102] Simon Schenk and Steffen Staab. Networked graphs: A declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In *Proceedings WWW-2008*, pages 585–594, Beijing, China, 2008. ACM Press.

[103] Roman Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, Vienna University of Technology, December 2006.

[104] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *13th International Conference on Database Theory (ICDT2010)*, Lausanne, Switzerland, March 2010.

[105] Michael Schneider, Jeremy Carroll, Ivan Herman, and Peter F. Patel-Schneider. W3C OWL 2 Web Ontology Language RDF-Based Semantics. W3C recommendation, W3C, October 2009. Available at `http://www.w3.org/TR/owl2-rdf-based-semantics/`.

[106] Michael Sintek and Stefan Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *1st International Semantic Web Conference*, pages 364–378, 2002.

[107] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL query for OWL-DL. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, Innsbruck, Austria, June 2007. CEUR-WS.org.

[108] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide. W3C recommendation, W3C, February 2004. Available at `http://www.w3.org/TR/owl-guide/`.

[109] SQL-99. Information Technology - Database Language SQL- Part 3: Call Level Interface (SQL/CLI). Technical Report INCITS/ISO/IEC 9075-3, INCITS/ISO/IEC, October 1999. Standard specification.

[110] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3:79–115, 2005.

[111] Giorgio Terracina, Nicola Leone, Vincenzino Lio, and Claudio Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory and Practice of Logic Programming*, 8(2):129–165, March 2008.

[112] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures, 2nd Edition. W3C Recommendation, W3C, October 2004. Available at `http://www.w3.org/TR/xmlschema-1/`.

[113] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using mapreduce. In *International Semantic Web Conference*, pages 634–649, 2009.

[114] Norman Walsh. RDF Twig: Accessing RDF Graphs in XSLT. Presented at Extreme Markup Languages (XML) 2003, Montreal, Canada. Available at `http://rdftwig.sourceforge.net/`.

[115] Jesse Weaver and James A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In *International Semantic Web Conference (ISWC2009)*, pages 682–697, 2009.

Accepted for publication in Knowledge and Information Systems (KAIS), Springer, 2010,
ISSN: 0219-1377 (print version), 0219-3116 (electronic version); cf.
http://www.springerlink.com/content/y3q6657333137683/)

A Semantical Framework for Hybrid Knowledge Bases

Jos de Bruijn[†] *David Pearce*[‡] *Axel Polleres*[§]
Agustín Valverde[¶]

[†] *Free University of Bozen-Bolzano, Bozen-Bolzano, Italy*
[‡] *Universidad Politécnica de Madrid, Spain*
[§] *DERI Galway, National University of Ireland, Galway, Ireland*
[¶] *Universidad de Málaga, Málaga, Spain*

Abstract

In the ongoing discussion about combining rules and Ontologies on the Semantic Web a recurring issue is how to combine first-order classical logic with nonmonotonic rule languages. Whereas several modular approaches to define a combined semantics for such hybrid knowledge bases focus mainly on decidability issues, we tackle the matter from a more general point of view. In this paper we show how Quantified Equilibrium Logic (QEL) can function as a unified framework which embraces classical logic as well as disjunctive logic programs under the (open) answer set semantics. In the proposed variant of QEL we relax the unique names assumption, which was present in earlier versions of QEL. Moreover, we show that this framework elegantly captures the existing modular approaches for hybrid knowledge bases in a unified way.

Keywords: Hybrid Knowledge Bases, Ontologies, Nonmonotonic Rules, Semantic Web, Logic Programming, Quantified Equilibrium Logic, Answer Set Programming

1 Introduction

In the current discussions on the Semantic Web architecture a recurring issue is how to combine a first-order classical theory formalising an ontology with a (possibly nonmonotonic) rule base. Nonmonotonic rule languages have received considerable attention and achieved maturity over the last few years especially due to the success of Answer Set Programming (ASP), a nonmonotonic, purely declarative logic programming and knowledge representation paradigm with many useful features such as aggregates, weak constraints and priorities, supported by efficient implementations (for an overview see [1]).

As a logical foundation for the answer set semantics and a tool for logical analysis in ASP, the system of Equilibrium Logic was presented in [24] and further developed in subsequent works (see [25] for an overview and references). The aim of this paper is to show how Equilibrium Logic can be used as a logical foundation for the combination of ASP and Ontologies.

In the quest to provide a formal underpinning for a nonmonotonic rules layer for the Semantic Web which can coexist in a semantically well-defined manner with the Ontology layer, various proposals for combining classical first-order logic with different variants of ASP have been presented in the literature.[1] We distinguish three kinds of approaches: At one end of the spectrum there are approaches which provide an entailment-based query interface to the Ontology in the bodies of ASP rules, resulting in a loose integration (e.g. [10, 9]). At the other end there are approaches which use a unifying nonmonotonic formalism to embed both the Ontology and the rule base (e.g. [4, 23]), resulting in a tight coupling. Hybrid approaches (e.g. [29, 30, 31, 16]) fall between these extremes. Common to hybrid approaches is the definition of a modular semantics based on classical first-order models, on the one hand, and stable models – often, more generally, referred to as answer sets[2] – on the other hand. Additionally, they require several syntactical restrictions on the use of classical predicates within rules, typically driven by considerations upon retaining decidability of reasoning tasks such as knowledge base satisfiability and predicate subsumption. With further restrictions of the classical part to decidable Description Logics (DLs), these semantics support straightforward implementation using existing DL reasoners and ASP engines, in a modular fashion. In this paper, we focus on such hybrid approaches, but from a more general point of view.

Example 1 Consider a hybrid knowledge base consisting of a classical theory $\mathcal{T}$:

$$\begin{array}{l}\forall x.PERSON(x) \rightarrow (AGENT(x) \wedge (\exists y.HAS\text{-}MOTHER(x,y))) \\ \forall x.(\exists y.HAS\text{-}MOTHER(x,y)) \rightarrow ANIMAL(x)\end{array}$$

which says that every $PERSON$ is an $AGENT$ and has some (unknown) mother, and everyone who has a mother is an $ANIMAL$, and a nonmonotonic logic program $\mathcal{P}$:

$$\begin{array}{l}PERSON(x) \leftarrow AGENT(x), \neg machine(x) \\ AGENT(DaveBowman)\end{array}$$

which says that $AGENT$s are by default $PERSON$s, unless *known* to be $machine$s, and $DaveBowman$ is an $AGENT$.

Using such a hybrid knowledge base consisting of $\mathcal{T}$ and $\mathcal{P}$, we intuitively would conclude that $PERSON(DaveBowman)$ holds since he is not known to be a $machine$, and furthermore we would conclude that $DaveBowman$ has some (unknown) mother, and thus $ANIMAL(DaveBowman)$. $\diamond$

We see two important shortcomings in current hybrid approaches:

[1]Most of these approaches focus on the Description Logics fragments of first-order logic underlying the Web Ontology Language OWL.

[2]"answer sets" denote the extension of stable models, which originally have only been defined for normal logic programs to more general logic programs such as disjunctive programs.

(1) Current approaches to hybrid knowledge bases differ not only in terms of syntactic restrictions, motivated by decidability considerations, but also in the way they deal with more fundamental issues which arise when classical logic meets ASP, such as the domain closure and unique names assumptions.[3] In particular, current proposals implicitly deal with these issues by either restricting the allowed models of the classical theory, or by using variants of the traditional answer set semantics which cater for open domains and non-unique names. So far, little effort has been spent in a comparing the approaches from a more general perspective. In this paper we aim to provide a generic semantic framework for hybrid knowledge bases that neither restricts models (e.g. to unique names) nor imposes syntactical restrictions driven by decidability concerns. (2) The semantics of current hybrid knowledge bases is defined in a modular fashion. This has the important advantage that algorithms for reasoning with this combination can be based on existing algorithms for DL and ASP satisfiability. A single underlying logic for hybrid knowledge bases which, for example, allows to capture notions of equivalence between combined knowledge bases in a standard way, is lacking though.

Our main contribution with this paper is twofold. First, we survey and compare different (extensions of the) answer set semantics, as well as the existing approaches to hybrid knowledge bases, all of which define nonmonotonic models in a modular fashion. Second, we propose to use Quantified Equilibrium Logic (QEL) as a unified logical foundation for hybrid knowledge bases: As it turns out, the equilibrium models of the combined knowledge base coincide exactly with the modular nonmonotonic models for all approaches we are aware of [29, 30, 31, 16].

The remainder of this paper is structured as follows: Section 2 recalls some basics of classical first-order logic. Section 3 reformulates different variants of the answer set semantics introduced in the literature using a common notation and points out correspondences and discrepancies between these variants. Next, definitions of hybrid knowledge bases from the literature are compared and generalised in Section 4. QEL and its relation to the different variants of ASP are clarified in Section 5. Section 6 describes an embedding of hybrid knowledge bases into QEL and establishes the correspondence between equilibrium models and nonmonotonic models of hybrid KBs. We discuss some immediate implications of our results in Section 7. In Section 8 we show how for finite knowledge bases an equivalent semantical characterisation can be given via a second-order operator NM. This behaves analogously to the operator SM used by Ferraris, Lee and Lifschitz [12] to define the stable models of a first-order sentence, except that its minimisation condition applies only to the non-classical predicates. In Section 9 we discuss an application of the previous results: we propose a definition of strong equivalence for knowledge bases sharing a common structural language and show how this notion can be captured by deduction in the (monotonic) logic of here-and-there. These two Sections (9 and 8) particularly contain mostly new material which has not yet been presented in the conference version [5] of this article. We conclude with a discussion of further related approaches and an outlook to future work in Section 10.

[3]See [3] for a more in-depth discussion of these issues.

2 First-Order Logic (FOL)

A *function-free first-order language* $\mathcal{L} = \langle C, P \rangle$ with equality consists of disjoint sets of constant and predicate symbols C and P. Moreover, we assume a fixed countably infinite set of variables, the symbols '$\rightarrow$', '$\vee$', '$\wedge$', '$\neg$', '$\exists$', '$\forall$', and auxiliary parentheses '(','$)$'. Each predicate symbol $p \in P$ has an assigned arity $ar(p)$. Atoms and formulas are constructed as usual. Closed formulas, or *sentences*, are those where each variable is bound by some quantifier. A *theory* $\mathcal{T}$ is a set of sentences. Variable-free atoms, formulas, or theories are also called *ground*. If D is a non-empty set, we denote by $At_D(C, P)$ the set of ground atoms constructible from $\mathcal{L}' = \langle C \cup D, P \rangle$.

Given a first-order language $\mathcal{L}$, an $\mathcal{L}$-structure consists of a pair $\mathcal{I} = \langle U, I \rangle$, where the *universe* $U = (D, \sigma)$ (sometimes called pre-interpretation) consists of a non-empty domain D and a function $\sigma\colon C \cup D \rightarrow D$ which assigns a domain value to each constant such that $\sigma(d) = d$ for every $d \in D$. For tuples we write $\sigma(\vec{t}) = (\sigma(d_1), \ldots, \sigma(d_n))$. We call $d \in D$ an *unnamed* individual if there is no $c \in C$ such that $\sigma(c) = d$. The function I assigns a relation $p^I \subseteq D^n$ to each n-ary predicate symbol $p \in P$ and is called the *$\mathcal{L}$-interpretation over D* . The designated binary predicate symbol eq, occasionally written '$=$' in infix notation, is assumed to be associated with the fixed interpretation function $eq^I = \{(d, d)\colon d \in D\}$. If $\mathcal{I}$ is an $\mathcal{L}'$-structure we denote by $\mathcal{I}|_{\mathcal{L}}$ the restriction of $\mathcal{I}$ to a sublanguage $\mathcal{L} \subseteq \mathcal{L}'$.

An $\mathcal{L}$-structure $\mathcal{I} = \langle U, I \rangle$ *satisfies* an atom $p(d_1, \ldots, d_n)$ of $At_D(C, P)$, written $\mathcal{I} \models p(d_1, \ldots, d_n)$, iff $(\sigma(d_1), \ldots, \sigma(d_n)) \in p^I$. This is extended as usual to sentences and theories. $\mathcal{I}$ is a *model* of an atom (sentence, theory, respectively) φ, written $\mathcal{I} \models \varphi$, if it satisfies φ. A theory $\mathcal{T}$ *entails* a sentence φ, written $\mathcal{T} \models \varphi$, if every model of $\mathcal{T}$ is also a model of φ. A theory is *consistent* if it has a model.

In the context of logic programs, the following assumptions often play a role: We say that the *parameter names assumption (PNA)* applies in case $\sigma|_C$ is surjective, i.e., there are no unnamed individuals in D; the *unique names assumption (UNA)* applies in case $\sigma|_C$ is injective; in case both the PNA and UNA apply, the *standard names assumption (SNA)* applies, i.e. $\sigma|_C$ is a bijection. In the following, we will speak about PNA-, UNA-, or SNA-structures, (or PNA-, UNA-, or SNA-models, respectively), depending on σ.

An $\mathcal{L}$-interpretation I over D can be seen as a subset of $At_D(C, P)$. So, we can define a subset relation for $\mathcal{L}$-structures $\mathcal{I}_1 = \langle (D, \sigma_1), I_1 \rangle$ and $\mathcal{I}_2 = \langle (D, \sigma_2), I_2 \rangle$ over the same domain by setting $\mathcal{I}_1 \subseteq \mathcal{I}_2$ if $I_1 \subseteq I_2$.[4] Whenever we speak about subset minimality of models/structures in the following, we thus mean minimality among all models/structures over the same domain.

[4]Note that this is not the substructure or submodel relation in classical model theory, which holds between a structure and its restriction to a subdomain.

3 Answer Set Semantics

In this paper we assume non-ground disjunctive logic programs with negation allowed in rule heads and bodies, interpreted under the answer set semantics [21].[5] A program $\mathcal{P}$ consists of a set of rules of the form

$$a_1 \vee a_2 \vee \ldots \vee a_k \vee \neg a_{k+1} \vee \ldots \vee \neg a_l \leftarrow b_1, \ldots, b_m, \neg b_{m+1}, \ldots, \neg b_n \qquad (1)$$

where a_i ($i \in \{1, \ldots, l\}$) and b_j ($j \in \{1, \ldots, n\}$) are atoms, called head (body, respectively) atoms of the rule, in a function-free first-order language $\mathcal{L} = \langle C, P \rangle$ without equality. By $C_{\mathcal{P}} \subseteq C$ we denote the set of constants which appear in $\mathcal{P}$. A rule with $k = l$ and $m = n$ is called *positive*. Rules where each variable appears in $b_1, \ldots, b_m$ are called *safe*. A program is *positive* (*safe*) if all its rules are positive (safe).

For the purposes of this paper, we give a slightly generalised definition of the common notion of the *grounding* of a program: The *grounding* $gr_U(\mathcal{P})$ of $\mathcal{P}$ wrt. a universe $U = (D, \sigma)$ denotes the set of all rules obtained as follows: For $r \in \mathcal{P}$, replace (i) each constant c appearing in r with $\sigma(c)$ and (ii) each variable with some element in D. Observe that thus $gr_U(\mathcal{P})$ is a ground program over the atoms in $At_D(C, P)$.

For a ground program $\mathcal{P}$ and first-order structure $\mathcal{I}$ the *reduct* $\mathcal{P}^{\mathcal{I}}$ consists of rules

$$a_1 \vee a_2 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m$$

obtained from all rules of the form (1) in $\mathcal{P}$ for which it holds that $\mathcal{I} \models a_i$ for all $k < i \leq l$ and $\mathcal{I} \not\models b_j$ for all $m < j \leq n$.

Answer set semantics is usually defined in terms of *Herbrand structures* over $\mathcal{L} = \langle C, P \rangle$. Herbrand structures have a fixed universe, the *Herbrand universe* $\mathcal{H} = (C, id)$, where id is the identity function. For a Herbrand structure $\mathcal{I} = \langle \mathcal{H}, I \rangle$, I can be viewed as a subset of the *Herbrand base*, $\mathcal{B}$, which consists of the ground atoms of $\mathcal{L}$. Note that by definition of $\mathcal{H}$, Herbrand structures are SNA-structures. A Herbrand structure $\mathcal{I}$ is an *answer set* [21] of $\mathcal{P}$ if $\mathcal{I}$ is subset minimal among the structures satisfying $gr_{\mathcal{H}}(\mathcal{P})^{\mathcal{I}}$. Two variations of this semantics, the open [15] and generalised open answer set [16] semantics, consider open domains, thereby relaxing the PNA. An *extended Herbrand structure* is a first-order structure based on a universe $U = (D, id)$, where $D \supseteq C$.

Definition 1 *A first-order $\mathcal{L}$-structure $\mathcal{I} = \langle U, I \rangle$ is called a* generalised open answer set *of $\mathcal{P}$ if $\mathcal{I}$ is subset minimal among the structures satisfying all rules in $gr_U(\mathcal{P})^{\mathcal{I}}$. If, additionally, $\mathcal{I}$ is an extended Herbrand structure, then $\mathcal{I}$ is an* open answer set *of $\mathcal{P}$.*

In the open answer set semantics the UNA applies. We have the following correspondence with the answer set semantics. First, as a straightforward consequence from the definitions, we can observe:

Proposition 1 *If $\mathcal{M}$ is an answer set of $\mathcal{P}$ then $\mathcal{M}$ is also an open answer set of $\mathcal{P}$.*

The converse does not hold in general:

[5]By $\neg$ we mean negation as failure and not classical, or strong negation, which is also sometimes considered in ASP.

Example 2 Consider $\mathcal{P} = \{p(a);\ ok \leftarrow \neg p(x);\ \leftarrow \neg ok\}$ over $\mathcal{L} = \langle\{a\}, \{p, ok\}\rangle$. We leave it as an exercise to the reader to show that $\mathcal{P}$ is inconsistent under the answer set semantics, but $\mathcal{M} = \langle(\{a, c_1\}, id), \{p(a), ok\}\rangle$ is an open answer set of $\mathcal{P}$. $\diamond$

Open answer set programs allow the use of the equality predicate '$=$' in the body of rules. However, since this definition of open answer sets adheres to the UNA, one could argue that equality in open answer set programming is purely syntactical. Positive equality predicates in rule bodies can thus be eliminated by simple preprocessing, applying unification. This is not the case for negative occurrences of equality, but, since the interpretation of equality is fixed, these can be eliminated during grounding.

An alternative approach to relax the UNA has been presented by Rosati in [30]: Instead of grounding with respect to U, programs are grounded with respect to the Herbrand universe $\mathcal{H} = (C, id)$, and minimality of the models of $gr_{\mathcal{H}}(\mathcal{P})^{\mathcal{I}}$ wrt. U is redefined: $\mathcal{I}\restriction_{\mathcal{H}} = \{p(\sigma(c_1), \ldots, \sigma(c_n))\colon p(c_1, \ldots, c_n) \in \mathcal{B}, \mathcal{I} \models p(c_1, \ldots, c_n)\}$, i.e., $\mathcal{I}\restriction_{\mathcal{H}}$ is the restriction of $\mathcal{I}$ to ground atoms of $\mathcal{B}$. Given $\mathcal{L}$-structures $\mathcal{I}_1 = (U_1, I_1)$ and $\mathcal{I}_2 = (U_2, I_2)$,[6] the relation $\mathcal{I}_1 \subseteq_{\mathcal{H}} \mathcal{I}_2$ holds if $\mathcal{I}_1\restriction_{\mathcal{H}} \subseteq \mathcal{I}_2\restriction_{\mathcal{H}}$.

Definition 2 *An $\mathcal{L}$-structure $\mathcal{I}$ is called a* generalised answer set *of $\mathcal{P}$ if $\mathcal{I}$ is $\subseteq_{\mathcal{H}}$-minimal among the structures satisfying all rules in $gr_{\mathcal{H}}(\mathcal{P})^{\mathcal{I}}$.*

The following Lemma (implicit in [14]) establishes that, for safe programs, all atoms of $At_D(C, P)$ satisfied in an open answer set of a safe program are ground atoms over $C_{\mathcal{P}}$:

Lemma 2 *Let $\mathcal{P}$ be a safe program over $\mathcal{L} = \langle C, P\rangle$ with $\mathcal{M} = \langle U, I\rangle$ a (generalised) open answer set over universe $U = (D, \sigma)$. Then, for any atom from $At_D(C, P)$ such that $\mathcal{M} \models p(d_1, \ldots, d_n)$, there exist $c_i \in C_{\mathcal{P}}$ such that $\sigma(c_i) = d_i$ for each $1 \leq i \leq n$.*

Proof: First, we observe that any atom $\mathcal{M} \models p(d_1, \ldots, d_n)$ must be derivable from a sequence of rules $(r_0; \ldots; r_l)$ in $gr_U(\mathcal{P})^{\mathcal{M}}$. We prove the lemma by induction over the length l of this sequence. $l = 0$: Assume $\mathcal{M} \models p(d_1, \ldots, d_n)$, then r_0 must be (by safety) a ground fact in $\mathcal{P}$ such that $p(\sigma(c_1), \ldots, \sigma(c_n)) = p(d_1, \ldots, d_n)$ and $c_1, \ldots, c_n \in C_{\mathcal{P}}$. As for the induction step, let $p(d_1, \ldots, d_n)$ be inferred by application of rule $r_l \in gr_U(\mathcal{P})^{\mathcal{M}}$. By safety, again each d_j either stems from a constant $c_j \in C_{\mathcal{P}}$ such that $\sigma(c_j) = d_j$ which appears in some true head atom of r_l or d_j also appears in a positive body atom $q(\ldots, d_j, \ldots)$ of r_l such that $\mathcal{M} \models q(\ldots, d_j, \ldots)$, derivable by $(r_0; \ldots; r_{l-1})$, which, by the induction hypothesis, proves the existence of a $c_j \in C_{\mathcal{P}}$ with $\sigma(c_j) = d_j$. $\Box$

From this Lemma, the following correspondence follows directly. Note that the answer sets and open answer sets of *safe* programs coincide as a direct consequence of Lemma 2:

Proposition 3 *$\mathcal{M}$ is an answer set of a* safe *program $\mathcal{P}$ if and only if $\mathcal{M}$ is an open answer set of $\mathcal{P}$.*

Similarly, on unsafe programs, generalised answer sets and generalised open answer sets do not necessarily coincide, as demonstrated by Example 2. However, the following correspondence follows straightforwardly from Lemma 2:

[6] Not necessarily over the same domain.

Proposition 4 *Given a safe program $\mathcal{P}$, $\mathcal{M}$ is a generalised open answer set of $\mathcal{P}$ if and only if M is a generalised answer set of $\mathcal{P}$.*

Proof:

($\Rightarrow$) Assume $\mathcal{M}$ is a generalised open answer set of $\mathcal{P}$. By Lemma 2 we know that rules in $gr_U(\mathcal{P})^{\mathcal{M}}$ involving unnamed individuals do not contribute to answer sets, since their body is always false. It follows that $\mathcal{M} = \mathcal{M}|_{\mathcal{H}}$ which in turn is a $\subseteq_{\mathcal{H}}$-minimal model of $gr_{\mathcal{H}}(\mathcal{P})^{\mathcal{M}}$. This follows from the observation that each rule in $gr_{\mathcal{H}}(\mathcal{P})^{\mathcal{M}}$ and its corresponding rules in $gr_U(\mathcal{P})^{\mathcal{M}}$ are satisfied under the same models.

($\Leftarrow$) Analogously.

$\square$

By similar arguments, generalised answer sets and generalised open answer sets coincide in case the parameter name assumption applies:

Proposition 5 *Let $\mathcal{M}$ be a PNA-structure. Then $\mathcal{M}$ is a generalised answer set of $\mathcal{P}$ if and only if $\mathcal{M}$ is a generalised open answer of $\mathcal{P}$.*

If the SNA applies, consistency with respect to all semantics introduced so far boils down to consistency under the original definition of answer sets:

Proposition 6 *A program $\mathcal{P}$ has an answer set if and only if $\mathcal{P}$ has a generalised open answer set under the SNA.*

Answer sets under SNA may differ from the original answer sets since also non-Herbrand structures are allowed. Further, we observe that there are programs which have generalised (open) answer sets but do not have (open) answer sets, even for safe programs, as shown by the following simple example:

Example 3 Consider $\mathcal{P} = \{p(a);\ \leftarrow \neg p(b)\}$ over $\mathcal{L} = \langle\{a, b\}, \{p\}\rangle$. $\mathcal{P}$ is ground, thus obviously safe. However, although $\mathcal{P}$ has a *generalised* (open) answer set – the reader may verify this by, for instance, considering the one-element universe $U = (\{d\}, \sigma)$, where $\sigma(a) = \sigma(b) = d$ – it is inconsistent under the open answer set semantics, i.e. the program does not have any open (non-genrealised) answer set. $\diamond$

4 Hybrid Knowledge Bases

We now turn to the concept of hybrid knowledge bases, which combine classical theories with the various notions of answer sets. We define a notion of hybrid knowledge bases which generalizes definitions in the literature [29, 30, 31, 16]. We then compare and discuss the differences between the various definitions. It turns out that the differences are mainly concerned with the notion of answer sets, and syntactical restrictions, but do not change the general semantics. This will allow us to base our embedding into Quantified Equilibrium Logic on a unified definition.

A *hybrid knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ over the function-free language $\mathcal{L} = \langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}}\rangle$ consists of a classical first-order theory $\mathcal{T}$ (also called the *structural* part of $\mathcal{K}$)

over the language $\mathcal{L}_\mathcal{T} = \langle C, P_\mathcal{T} \rangle$ and a program $\mathcal{P}$ (also called *rules* part of $\mathcal{K}$) over the language $\mathcal{L}$, where $P_\mathcal{T} \cap P_\mathcal{P} = \emptyset$, i.e. $\mathcal{T}$ and $\mathcal{P}$ share a single set of constants, and the predicate symbols allowed to be used in $\mathcal{P}$ are a superset of the predicate symbols in $\mathcal{L}_\mathcal{T}$. Intuitively, the predicates in $\mathcal{L}_\mathcal{T}$ are interpreted classically, whereas the predicates in $\mathcal{L}_\mathcal{P}$ are interpreted nonmonotonically under the (generalised open) answer set semantics. With $\mathcal{L}_\mathcal{P} = \langle C, P_\mathcal{P} \rangle$ we denote the restricted language of $\mathcal{P}$ to only the distinct predicates $P_\mathcal{P}$ which are not supposed to occur in $\mathcal{T}$.

We do not consider the alternative classical semantics defined in [29, 30, 31], as these are straightforward.

We define the *projection* of a ground program $\mathcal{P}$ with respect to an $\mathcal{L}$-structure $\mathcal{I} = \langle U, I \rangle$, denoted $\Pi(\mathcal{P}, \mathcal{I})$, as follows: for each rule $r \in \mathcal{P}$, r^Π is defined as:

1. $r^\Pi = \emptyset$ if there is a literal over $At_D(C, P_\mathcal{T})$ in the head of r of form $p(\vec{t})$ such that $p(\sigma(\vec{t})) \in I$ or of form $\neg p(\vec{t})$ with $p(\sigma(\vec{t})) \notin I$;

2. $r^\Pi = \emptyset$ if there is a literal over $At_D(C, P_\mathcal{T})$ in the body of r of form $p(\vec{t})$ such that $p(\sigma(\vec{t})) \notin I$ or of form $\neg p(\vec{t})$ such that $p(\sigma(\vec{t})) \in I$;

3. otherwise r^Π is the singleton set resulting from r by deleting all occurrences of literals from $\mathcal{L}_\mathcal{T}$,

and $\Pi(\mathcal{P}, \mathcal{I}) = \bigcup\{r^\Pi \colon r \in \mathcal{P}\}$. Intuitively, the projection "evaluates" all classical literals in $\mathcal{P}$ with respect to $\mathcal{I}$.

Definition 3 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be a hybrid knowledge base over the language $\mathcal{L} = \langle C, P_\mathcal{T} \cup P_\mathcal{P} \rangle$. An NM-model $\mathcal{M} = \langle U, I \rangle$ (with $U = (D, \sigma)$) of $\mathcal{K}$ is a first-order $\mathcal{L}$-structure such that $\mathcal{M}|_{\mathcal{L}_\mathcal{T}}$ is a model of $\mathcal{T}$ and $\mathcal{M}|_{\mathcal{L}_\mathcal{P}}$ is a generalised open answer set of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$.*

Analogous to first-order models, we speak about PNA-, UNA-, and SNA-NM-models.

Example 4 Consider the hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$, with $\mathcal{T}$ and $\mathcal{P}$ as in Example 1, with the capitalised predicates being predicates in $P_\mathcal{T}$. Now consider the interpretation $\mathcal{I} = \langle U, I \rangle$ (with $U = (D, \sigma)$) with $D = \{DaveBowman, k\}$, σ the identity function, and $I = \{AGENT(DaveBowman), HAS\text{-}MOTHER(DaveBowman, k),$ $ANIMAL(DaveBowman), machine(DaveBowman)\}$. Clearly, $\mathcal{I}|_{\mathcal{L}_\mathcal{T}}$ is a model of $\mathcal{T}$. The projection $\Pi(gr_U(\mathcal{P}), \mathcal{I})$ is

$$\leftarrow \neg machine(DaveBowman),$$

which does not have a stable model, and thus $\mathcal{I}$ is not an NM-model of $\mathcal{K}$. In fact, the logic program $\mathcal{P}$ ensures that an interpretation cannot be an NM-model of $\mathcal{K}$ if there is an $AGENT$ which is neither a $PERSON$ nor known (by conclusions from $\mathcal{P}$) to be a $machine$. It is easy to verify that, for any NM-model of $\mathcal{K}$, the atoms $AGENT(DaveBowman)$, $PERSON(DaveBowman)$, and $ANIMAL(DaveBowman)$ must be true, and are thus entailed by $\mathcal{K}$. The latter cannot be derived from neither $\mathcal{T}$ nor $\mathcal{P}$ individually. $\diamond$

4.1 r-hybrid KBs

We now proceed to compare our definition of NM-models with the various definitions in the literature. The first kind of hybrid knowledge base we consider was introduced by Rosati in [29] (and extended in [31] under the name $\mathcal{DL}$+log), and was labeled *r-hybrid* knowledge base. Syntactically, *r-hybrid* KBs do not allow negated atoms in rule heads, i.e. for rules of the form (1) $l = k$, and do not allow atoms from $\mathcal{L}_\mathcal{T}$ to occur negatively in the rule body.[7] Moreover, in [29], Rosati deploys a restriction which is stronger than standard safety: each variable must appear in at least one positive body atom with a predicate from $\mathcal{L}_\mathcal{P}$. We call this condition *$\mathcal{L}_\mathcal{P}$-safe* in the remainder. In [31] this condition is relaxed to *weak $\mathcal{L}_\mathcal{P}$-safety*: there is no special safety restriction for variables which occur only in body atoms from $P_\mathcal{T}$.

Semantically, Rosati assumes (an infinite number of) standard names, i.e. C is countably infinite, and normal answer sets, in his version of NM-models:

Definition 4 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be an r-hybrid knowledge base, over the language $\mathcal{L} = \langle C, P_\mathcal{T} \cup P_\mathcal{P} \rangle$, where C is countably infinite, and $\mathcal{P}$ is a (weak) $\mathcal{L}_\mathcal{P}$-safe program. An* r-NM-model *$\mathcal{M} = \langle U, I \rangle$ of $\mathcal{K}$ is a first-order $\mathcal{L}$-SNA-structure such that $\mathcal{M}|_{\mathcal{L}_\mathcal{T}}$ is a model of $\mathcal{T}$ and $\mathcal{M}|_{\mathcal{L}_\mathcal{P}}$ is an answer set of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$.*

In view of the (weak) $\mathcal{L}_\mathcal{P}$-safety condition, we observe that r-NM-model existence coincides with SNA-NM-model existence on r-hybrid knowledge bases, by Lemma 2 and Proposition 6.

The syntactic restrictions in r-hybrid knowledge bases guarantee decidability of the satisfiability problem in case satisfiability (in case of $\mathcal{L}_\mathcal{P}$-safety) or conjunctive query containment (in case of weak $\mathcal{L}_\mathcal{P}$-safety) in $\mathcal{T}$ is decidable. Rosati [29, 31] presents sound and complete algorithms for both cases.

4.2 r$^+$-hybrid KBs

In [30], Rosati relaxes the UNA for what we will call here *r$^+$-hybrid* knowledge bases. In this variant the $\mathcal{L}_\mathcal{P}$-safety restriction is kept but generalised answer sets under arbitrary interpretations are considered:

Definition 5 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be an r$^+$-hybrid knowledge base consisting of a theory $\mathcal{T}$ and an $\mathcal{L}_\mathcal{P}$-safe program $\mathcal{P}$. An* r$^+$-NM-model*, $\mathcal{M} = \langle U, I \rangle$ of $\mathcal{K}$ is a first-order $\mathcal{L}$-structure such that $\mathcal{M}|_{\mathcal{L}_\mathcal{T}}$ is a model of $\mathcal{T}$ and $\mathcal{M}|_{\mathcal{L}_\mathcal{P}}$ is a generalised answer set of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$.*

$\mathcal{L}_\mathcal{P}$-safety guarantees safety of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$. Thus, by Proposition 3, we can conclude that r$^+$-NM-models coincide with NM-models on r-hybrid knowledge bases. The relaxation of the UNA does not affect decidability

[7] Note that by projection, negation of predicates from $P_\mathcal{T}$ is treated classically, whereas negation of predicates from $P_\mathcal{P}$ is treated nonmonotonically. This might be considered unintuitive and therefore a reason why Rosati disallows structural predicates to occur negated. The negative occurrence of classical predicates in the body is equivalent to the positive occurrence of the predicate in the head.

	SNA	variables	disjunctive rule heads	negated $\mathcal{L}_\mathcal{T}$ atoms
r-hybrid	yes	$\mathcal{L}_\mathcal{P}$-safe	pos. only	no
r^+-hybrid	no	$\mathcal{L}_\mathcal{P}$-safe	pos. only	no
r_w-hybrid	yes	weak $\mathcal{L}_\mathcal{P}$-safe	pos. only	no
g-hybrid	no	guarded	neg. allowed*	yes

* g-hybrid allows negation in the head but at most one positive head atom

Table 1: Different variants of hybrid KBs

4.3 g-hybrid KBs

G-hybrid knowledge bases [16] allow a different form of rules in the program. In order to regain decidability, rules are not required to be safe, but they are required to be *guarded* (hence the 'g' in g-hybrid): All variables in a rule are required to occur in a single positive body atom, the *guard*, with the exception that unsafe choice rules of the form

$$p(c_1, \ldots, c_n) \vee \neg p(c_1, \ldots, c_n) \leftarrow$$

are allowed. Moreover, disjunction in rule heads is limited to at most one positive atom, i.e. for rules of the form (1) we have that $k \leq 1$, but an arbitrary number of negated head atoms is allowed. Another significant difference is that, as opposed to the approaches based on r-hybrid KBs, negative structural predicates are allowed in the rules part within g-hybrid knowledge bases (see also Footnote 7). The definition of NM-models in [16] coincides precisely with our Definition 3.

Table 4.3 summarises the different versions of hybrid knowledge bases introduced in the literature.

5 Quantified Equilibrium Logic (QEL)

Equilibrium logic for propositional theories and logic programs was presented in [24] as a foundation for answer set semantics, and extended to the first-order case in [26], as well as, in slightly more general, modified form, in [27]. For a survey of the main properties of equilibrium logic, see [25]. Usually in quantified equilibrium logic we consider a full first-order language allowing function symbols and we include a second, strong negation operator as occurs in several ASP dialects. For the present purpose of drawing comparisons with approaches to hybrid knowledge bases, it will suffice to consider the function-free language with a single negation symbol, '$\neg$'. In particular, we shall work with a quantified version of the logic HT of *here-and-there*. In other respects we follow the treatment of [27].

5.1 General Structures for Quantified Here-and-There Logic

As before, we consider a function-free first order language $\mathcal{L} = \langle C, P \rangle$ built over a set of *constant* symbols, C, and a set of *predicate* symbols, P. The sets of $\mathcal{L}$-

formulas, $\mathcal{L}$-sentences and atomic $\mathcal{L}$-sentences are defined in the usual way. Again, we only work with *sentences*, and, as in Section 2, by an $\mathcal{L}$-interpretation I over a set D we mean a subset I of $At_D(C, P)$. A *here-and-there* $\mathcal{L}$-structure with static domains, or $\mathbf{QHT}^s(\mathcal{L})$*-structure*, is a tuple $\mathcal{M} = \langle (D, \sigma), I_h, I_t \rangle$ where $\langle (D, \sigma), I_h \rangle$ and $\langle (D, \sigma), I_t \rangle$ are $\mathcal{L}$-structures such that $I_h \subseteq I_t$.

We can think of $\mathcal{M}$ as a structure similar to a first-order classical model, but having two parts, or components, h and t that correspond to two different points or "worlds", 'here' and 'there', in the sense of Kripke semantics for intuitionistic logic [32], where the worlds are ordered by $h \leq t$. At each world $w \in \{h, t\}$ one verifies a set of atoms I_w in the expanded language for the domain D. We call the model static, since, in contrast to say intuitionistic logic, the same domain serves each of the worlds.[8] Since $h \leq t$, whatever is verified at h remains true at t. The satisfaction relation for $\mathcal{M}$ is defined so as to reflect the two different components, so we write $\mathcal{M}, w \models \varphi$ to denote that φ is true in $\mathcal{M}$ with respect to the w component. Evidently we should require that an atomic sentence is true at w just in case it belongs to the w-interpretation. Formally, if $p(t_1, \ldots, t_n) \in \mathrm{At}_D$ then

$$\mathcal{M}, w \models p(t_1, \ldots, t_n) \quad \text{iff} \quad p(\sigma(t_1), \ldots, \sigma(t_n)) \in I_w. \tag{2}$$

Then $\models$ is extended recursively as follows[9]:

- $\mathcal{M}, w \models \varphi \wedge \psi$ iff $\mathcal{M}, w \models \varphi$ and $\mathcal{M}, w \models \psi$.
- $\mathcal{M}, w \models \varphi \vee \psi$ iff $\mathcal{M}, w \models \varphi$ or $\mathcal{M}, w \models \psi$.
- $\mathcal{M}, t \models \varphi \to \psi$ iff $\mathcal{M}, t \not\models \varphi$ or $\mathcal{M}, t \models \psi$.
- $\mathcal{M}, h \models \varphi \to \psi$ iff $\mathcal{M}, t \models \varphi \to \psi$ and $\mathcal{M}, h \not\models \varphi$ or $\mathcal{M}, h \models \psi$.
- $\mathcal{M}, w \models \neg\varphi$ iff $\mathcal{M}, t \not\models \varphi$.
- $\mathcal{M}, t \models \forall x \varphi(x)$ iff $\mathcal{M}, t \models \varphi(d)$ for all $d \in D$.
- $\mathcal{M}, h \models \forall x \varphi(x)$ iff $\mathcal{M}, t \models \forall x \varphi(x)$ and $\mathcal{M}, h \models \varphi(d)$ for all $d \in D$.
- $\mathcal{M}, w \models \exists x \varphi(x)$ iff $\mathcal{M}, w \models \varphi(d)$ for some $d \in D$.

Truth of a sentence in a model is defined as follows: $\mathcal{M} \models \varphi$ iff $\mathcal{M}, w \models \varphi$ for each $w \in \{h, t\}$. A sentence φ is valid if it is true in all models, denoted by $\models \varphi$. A sentence φ is a consequence of a set of sentences Γ, denoted $\Gamma \models \varphi$, if every model of Γ is a model of φ. In a model $\mathcal{M}$ we often use the symbols H and T, possibly with subscripts, to denote the interpretations I_h and I_t respectively; so, an $\mathcal{L}$-structure may be written in the form $\langle U, H, T \rangle$, where $U = (D, \sigma)$.

The resulting logic is called *Quantified Here-and-There Logic with static domains*, denoted by $\mathbf{QHT}^s$. In terms of satisfiability and validity this logic is equivalent to

[8]Alternatively it is quite common to speak of a logic with *constant* domains. However this is slightly ambiguous since it might suggest that the domain is composed only of constants, which is not intended here.

[9]The reader may easily check that the following correspond exactly to the usual Kripke semantics for intuitionistic logic given our assumptions about the two worlds h and t and the single domain D, see e.g. [32]

the logic introduced before in [26]. By $\mathbf{QHT}^s_=$ we denote the version of QEL with equality. The equality predicate in $\mathbf{QHT}^s_=$ is interpreted as the actual equality in both worlds, ie $\mathcal{M}, w \models t_1 = t_2$ iff $\sigma(t_1) = \sigma(t_2)$.

The logic $\mathbf{QHT}^s_=$ can be axiomatised as follows. Let **INT**$^=$ denote first-order intuitionistic logic [32] with the usual axioms for equality:

$$x = x,$$
$$x = y \rightarrow (F(x) \rightarrow F(y)),$$

for every formula $F(x)$ such that y is substitutable for x in $F(x)$. To this we add the axiom of Hosoi

$$\alpha \vee (\neg\beta \vee (\alpha \rightarrow \beta)),$$

which determines 2-element here-and-there models in the propositional case, and the axiom SQHT (static quantified here-and-there):

$$\exists x(F(x) \rightarrow \forall x F(x)).$$

Lastly we add the "decidable equality" axiom:

$$x = y \vee x \neq y.$$

For a completeness proof for $\mathbf{QHT}^s_=$, see [20].

As usual in first order logic, satisfiability and validity are independent from the language. If $\mathcal{M} = \langle (D, \sigma), H, T\rangle$ is an $\mathbf{QHT}^s_=(\mathcal{L}')$-structure and $\mathcal{L} \subset \mathcal{L}'$, we denote by $\mathcal{M}|_{\mathcal{L}}$ the restriction of $\mathcal{M}$ to the sublanguage $\mathcal{L}$: $\mathcal{M}|_{\mathcal{L}} = \langle (D, \sigma|_{\mathcal{L}}), H|_{\mathcal{L}}, T|_{\mathcal{L}}\rangle$.

Proposition 7 *Suppose that $\mathcal{L}' \supset \mathcal{L}$, Γ is a theory in $\mathcal{L}$ and $\mathcal{M}$ is an $\mathcal{L}'$-structure such $\mathcal{M} \models \Gamma$. Then $\mathcal{M}|_{\mathcal{L}}$ is a model of Γ in $\mathbf{QHT}^s_=(\mathcal{L})$.*

Proposition 8 *Suppose that $\mathcal{L}' \supset \mathcal{L}$ and $\varphi \in \mathcal{L}$. Then φ is valid (resp. satisfiable) in $\mathbf{QHT}^s_=(\mathcal{L})$ if and only if is valid (resp. satisfiable) in $\mathbf{QHT}^s_=(\mathcal{L}')$.*

Analogous to the case of classical models we can define special kinds of $\mathbf{QHT}^s$ (resp. $\mathbf{QHT}^s_=$) models. Let $\mathcal{M} = \langle (D, \sigma), H, T\rangle$ be an $\mathcal{L}$-structure that is a model of a universal theory T. Then, we call $\mathcal{M}$ a PNA-, UNA-, or SNA-model if the restriction of σ to constants in $\mathcal{C}$ is surjective, injective or bijective, respectively.

5.2 Equilibrium Models

As in the propositional case, quantified equilibrium logic is based on a suitable notion of minimal model.

Definition 6 *Among $\mathbf{QHT}^s_=(\mathcal{L})$-structures we define the order $\trianglelefteq$ as: $\langle (D, \sigma), H, T\rangle \trianglelefteq \langle (D', \sigma'), H', T'\rangle$ if $D = D'$, $\sigma = \sigma'$, $T = T'$ and $H \subseteq H'$. If the subset relation is strict, we write '$\triangleleft$'.*

Definition 7 *Let Γ be a set of sentences and $\mathcal{M} = \langle (D, \sigma), H, T\rangle$ a model of Γ.*

1. *$\mathcal{M}$ is said to be* total *if $H = T$.*
2. *$\mathcal{M}$ is said to be an* equilibrium *model of Γ (for short, we say: "$\mathcal{M}$ is in equilibrium") if it is minimal under $\unlhd$ among models of Γ, and it is total.*

Notice that a total $\mathbf{QHT}^s_=$ model of a theory Γ is equivalent to a classical first order model of Γ.

Proposition 9 *Let Γ be a theory in $\mathcal{L}$ and $\mathcal{M}$ an equilibrium model of Γ in $\mathbf{QHT}^s_=(\mathcal{L}')$ with $\mathcal{L}' \supset \mathcal{L}$. Then $\mathcal{M}|_{\mathcal{L}}$ is an equilibrium model of Γ in $\mathbf{QHT}^s_=(\mathcal{L})$.*

5.3 Relation to Answer Sets

The above version of QEL is described in more detail in [27]. If we assume all models are UNA-models, we obtain the version of QEL found in [26]. There, the relation of QEL to (ordinary) answer sets for logic programs with variables was established (in [26, Corollary 7.7]). For the present version of QEL the correspondence can be described as follows.

Proposition 10 ([27]) *Let Γ be a universal theory in $\mathcal{L} = \langle C, P\rangle$. Let $\langle U, T, T\rangle$ be a total $\mathbf{QHT}^s_=$ model of Γ. Then $\langle U, T, T\rangle$ is an equilibrium model of Γ iff $\langle T, T\rangle$ is a propositional equilibrium model of $gr_U(\Gamma)$.*

By convention, when $\mathcal{P}$ is a logic program with variables we consider the models and equilibrium models of its universal closure expressed as a set of logical formulas. So, from Proposition 10 we obtain:

Corollary 11 *Let $\mathcal{P}$ be a logic program. A total $\mathbf{QHT}^s_=$ model $\langle U, T, T\rangle$ of $\mathcal{P}$ is an equilibrium model of $\mathcal{P}$ iff it is a generalised open answer set of $\mathcal{P}$.*

Proof: It is well-known that for propositional programs equilibrium models coincide with answer sets [24]. Using Proposition 10 and Definition 4 for generalised open answer sets, the result follows. □

6 Relation between Hybrid KBs and QEL

In this section we show how equilibrium models for hybrid knowledge bases relate to the NM models defined earlier and we show that QEL captures the various approaches to the semantics of hybrid KBs in the literature [29, 30, 31, 16].

Given a hybrid KB $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ we call $\mathcal{T} \cup \mathcal{P} \cup st(\mathcal{T})$ the *stable closure* of $\mathcal{K}$, where $st(\mathcal{T}) = \{\forall x(p(x) \vee \neg p(x)) \colon p \in \mathcal{L}_{\mathcal{T}}\}$.[10] From now on, unless otherwise clear from context, the symbol '$\models$' denotes the truth relation for $\mathbf{QHT}^s_=$. Given a ground program $\mathcal{P}$ and an $\mathcal{L}$-structure $\mathcal{M} = \langle U, H, T\rangle$, the *projection* $\Pi(\mathcal{P}, \mathcal{M})$ is understood to be defined relative to the component T of $\mathcal{M}$.

[10]Evidently $\mathcal{T}$ becomes *stable* in $\mathcal{K}$ in the sense that $\forall \varphi \in \mathcal{T}$, $st(\mathcal{T}) \models \neg\neg\varphi \rightarrow \varphi$. The terminology is drawn from intuitionistic logic and mathematics.

Lemma 12 *Let $\mathcal{M} = \langle U, H, T\rangle$ be a $\mathbf{QHT}^s_=$-model of $\mathcal{T} \cup st(\mathcal{T})$. Then $\mathcal{M} \models \mathcal{P}$ iff $\mathcal{M}|_{\mathcal{L}_\mathcal{P}} \models \Pi(gr_U(\mathcal{P}), \mathcal{M})$.*

Proof: By the hypothesis $\mathcal{M} \models \{\forall x(p(x) \vee \neg p(x)) \colon p \in \mathcal{L}_\mathcal{T}\}$. It follows that $H|_{\mathcal{L}_\mathcal{T}} = T|_{\mathcal{L}_\mathcal{T}}$. Consider any $r \in \mathcal{P}$, such that $r^\Pi \neq \emptyset$. Then there are four cases to consider. (i) r has the form $\alpha \rightarrow \beta \vee p(t)$, $p(t) \in \mathcal{L}_\mathcal{T}$ and $p(\sigma(t)) \notin T$, so $\mathcal{M} \models \neg p(t)$. W.l.o.g. assume that $\alpha, \beta \in \mathcal{L}_\mathcal{P}$, so $r^\Pi = \alpha \rightarrow \beta$ and

$$\mathcal{M} \models r \Leftrightarrow \mathcal{M} \models r^\Pi \Leftrightarrow \mathcal{M}|_{\mathcal{L}_\mathcal{P}} \models r^\Pi \tag{3}$$

by the semantics for $\mathbf{QHT}^s_=$ and Theorem 7. (ii) r has the form $\alpha \rightarrow \beta \vee \neg p(t)$, where $p(\sigma(t)) \in T$; so $p(\sigma(t)) \in H$ and $\mathcal{M} \models p(t)$. Again it is easy to see that (3) holds. Case (iii): r has the form $\alpha \wedge p(t) \rightarrow \beta$ and $p(\sigma(t)) \in H, T$, so $\mathcal{M} \models p(t)$. Case (iv): r has the form $\alpha \wedge \neg p(t) \rightarrow \beta$ and $\mathcal{M} \models \neg p(t)$. Clearly for these two cases (3) holds as well. It follows that if $\mathcal{M} \models \mathcal{P}$ then $\mathcal{M}|_{\mathcal{L}_\mathcal{P}} \models \Pi(gr_U(\mathcal{P}), \mathcal{M})$.

To check the converse condition we need now only examine the cases where $r^\Pi = \emptyset$. Suppose this arises because $p(\sigma(t)) \in H, T$, so $\mathcal{M} \models p(t)$. Now, if $p(t)$ is in the head of r, clearly $\mathcal{M} \models r$. Similarly if $\neg p(t)$ is in the body of r, by the semantics $\mathcal{M} \models r$. The cases where $p(\sigma(t)) \notin T$ are analogous and left to the reader. Consequently if $\mathcal{M}|_{\mathcal{L}_\mathcal{P}} \models \Pi(gr_U(\mathcal{P}), \mathcal{M})$, then $\mathcal{M} \models \mathcal{P}$. □

We now state the relation between equilibrium models and NM-models.

Theorem 13 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be a hybrid KB. $\mathcal{M} = \langle U, T, T\rangle$ is an equilibrium model of the stable closure of $\mathcal{K}$ if and only if $\langle U, T\rangle$ is an NM-model of $\mathcal{K}$.*

Proof: Assume the hypothesis and suppose that $\mathcal{M}$ is in equilibrium. Since $\mathcal{T}$ contains only predicates from $\mathcal{L}_\mathcal{T}$ and $\mathcal{M} \models \mathcal{T} \cup st(\mathcal{T})$, evidently

$$\mathcal{M}|_{\mathcal{L}_\mathcal{T}} \models \mathcal{T} \cup st(\mathcal{T}) \tag{4}$$

and so in particular $(U, \mathcal{M}|_{\mathcal{L}_\mathcal{T}})$ is a model of $\mathcal{T}$. By Lemma 12,

$$\mathcal{M} \models \mathcal{P} \Leftrightarrow \mathcal{M}|_{\mathcal{L}_\mathcal{P}} \models \Pi(gr_U(\mathcal{P}), \mathcal{M}). \tag{5}$$

We claim (i) that $\mathcal{M}|_{\mathcal{L}_\mathcal{P}}$ is an equilibrium model of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$. If not, there is a model $\mathcal{M}' = \langle H', T'\rangle$ with $H' \subset T' = T|_{\mathcal{L}_\mathcal{P}}$ and $\mathcal{M}' \models \Pi(gr_U(\mathcal{P}), \mathcal{M})$. Lift $(U, \mathcal{M}')$ to a (first order) $\mathcal{L}$-structure $\mathcal{N}$ by interpreting each $p \in \mathcal{L}_\mathcal{T}$ according to $\mathcal{M}$. So $\mathcal{N}|_{\mathcal{L}_\mathcal{T}} = \mathcal{M}|_{\mathcal{L}_\mathcal{T}}$ and by(4) clearly $\mathcal{N} \models \mathcal{T} \cup st(\mathcal{T})$. Moreover, by Lemma 12 $\mathcal{N} \models \mathcal{P}$ and by assumption $\mathcal{N} \lhd \mathcal{M}$, contradicting the assumption that $\mathcal{M}$ is an equilibrium model of $\mathcal{T} \cup st(\mathcal{T}) \cup \mathcal{P}$. This establishes (i). Lastly, we note that since $\langle T|_{\mathcal{L}_\mathcal{P}}, T|_{\mathcal{L}_\mathcal{P}}\rangle$ is an equilibrium model of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$, $\mathcal{M}|_{\mathcal{L}_\mathcal{P}}$ is a generalised open answer set of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$ by Corollary 11, so that $\mathcal{M} = \langle U, T, T\rangle$ is an NM-model of $\mathcal{K}$.

For the converse direction, assume the hypothesis but suppose that $\mathcal{M}$ is not in equilibrium. Then there is a model $\mathcal{M}' = \langle U, H, T\rangle$ of $\mathcal{T} \cup st(\mathcal{T}) \cup \mathcal{P}$, with $H \subset T$. Since $\mathcal{M}' \models \mathcal{P}$ we can apply Lemma 12 to conclude that $\mathcal{M}'|_{\mathcal{L}_\mathcal{P}} \models \Pi(gr_U(\mathcal{P}), \mathcal{M}')$. But clearly

$$\Pi(gr_U(\mathcal{P}), \mathcal{M}') = \Pi(gr_U(\mathcal{P}), \mathcal{M}).$$

However, since evidently $\mathcal{M}'|_{\mathcal{L_T}} = \mathcal{M}|_{\mathcal{L_T}}$, thus $\mathcal{M}'|_{\mathcal{L_P}} \lhd \mathcal{M}|_{\mathcal{L_P}}$, so this shows that $\mathcal{M}|_{\mathcal{L_P}}$ is not an equilibrium model of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$ and therefore $T|_{\mathcal{L_P}}$ is not an answer set of $\Pi(gr_U(\mathcal{P}), \mathcal{M})$ and $\mathcal{M}$ is not an NM- model of $\mathcal{K}$. □

This establishes the main theorem relating to the various special types of hybrid KBs discussed earlier.

Theorem 14 (Main Theorem) *(i) Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be a g-hybrid (resp. an r$^+$-hybrid) knowledge base. $\mathcal{M} = \langle U, T, T\rangle$ is an equilibrium model of the stable closure of $\mathcal{K}$ if and only if $\langle U, T\rangle$ is an NM-model (resp. r$^+$-NM-model) of $\mathcal{K}$.*
(ii) Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be an r-hybrid knowledge base. Let $\mathcal{M} = \langle U, T, T\rangle$ be an Herbrand model of the stable closure of $\mathcal{K}$. Then $\mathcal{M}$ is in equilibrium in the sense of [26] if and only if $\langle U, T\rangle$ is an r-NM-model of $\mathcal{K}$.

Example 5 Consider again the hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$, with $\mathcal{T}$ and $\mathcal{P}$ as in Example 1. The stable closure of $\mathcal{K}$, $st(\mathcal{K}) = \mathcal{T} \cup st(\mathcal{T}) \cup \mathcal{P}$ is

$$
\begin{array}{l}
\forall x.PERSON(x) \to (AGENT(x) \land (\exists y.HAS\text{-}MOTHER(x, y))) \\
\forall x.(\exists y.HAS\text{-}MOTHER(x, y)) \to ANIMAL(x) \\
\forall x.PERSON(x) \lor \neg PERSON(x) \\
\forall x.AGENT(x) \lor \neg AGENT(x) \\
\forall x.ANIMAL(x) \lor \neg ANIMAL(x) \\
\forall x, y.HAS\text{-}MOTHER(x, y) \lor \neg HAS\text{-}MOTHER(x, y) \\
\forall x.AGENT(x) \land \neg machine(x) \to PERSON(x) \\
AGENT(DaveBowman)
\end{array}
$$

Consider the total HT-model $\mathcal{M}_{HT} = \langle U, I, I\rangle$ of $st(\mathcal{K})$, with U, I as in Example 4. $\mathcal{M}_{HT}$ is *not* an equilibrium model of $st(\mathcal{K})$, since $\mathcal{M}_{HT}$ is not minimal among all models: $\langle U, I', I\rangle$, with $I' = I \backslash \{machine(DaveBowman)\}$, is a model of $st(\mathcal{K})$. Furthermore, it is easy to verify that $\langle U, I', I'\rangle$ is not a model of $st(\mathcal{K})$.

Now, consider the total HT-model $\mathcal{M}'_{HT} = \langle U, M, M\rangle$, with U as before, and

$$
\begin{aligned}
M = \{&AGENT(DaveBowman), PERSON(DaveBowman), \\
&ANIMAL(DaveBowman), HAS\text{-}NAME(DaveBowman, k)\}.
\end{aligned}
$$

$\mathcal{M}'_{HT}$ is an equilibrium model of $st(\mathcal{K})$. Indeed, consider any $M' \subset M$. It is easy to verify that $\langle U, M', M\rangle$ is not a model of $st(\mathcal{K})$. ◇

7 Discussion

We have seen that quantified equilibrium logic captures three of the main approaches to integrating classical, first-order or DL knowledge bases with nonmonotonic rules under the answer set semantics, in a modular, hybrid approach. However, QEL has a quite distinct flavor from those of r-hybrid, r$^+$-hybrid and g-hybrid KBs. Each of these hybrid approaches has a semantics composed of two different components: a classical model on the one hand and an answer set on the other. Integration is achieved by the fact that the classical model serves as a pre-processing tool for the rule base. The style of QEL is different. There is one semantics and one kind of model that covers both types of knowledge. There is no need for any pre-processing of the rule base. In this sense, the integration is more far-reaching. The only distinction we make is that for that part of the knowledge base considered to be classical and monotonic we add a stability condition to obtain the intended interpretation.

There are other features of the approach using QEL that are worth highlighting. First, it is based on a simple minimal model semantics in a known non-classical logic, actually a quantified version of Gödel's 3-valued logic. No reducts are involved and, consequently, the equilibrium construction applies directly to arbitrary first-order theories. The rule part $\mathcal{P}$ of a knowledge base might therefore comprise, say, a nested logic program, where the heads and bodies of rules may be arbitrary boolean formulas, or perhaps rules permitting nestings of the implication connective. While answer sets have recently been defined for such general formulas, more work would be needed to provide integration in a hybrid KB setting.[11] Evidently QEL in the general case is undecidable, so for extensions of the rule language syntax for practical applications one may wish to study restrictions analogous to safety or guardedness. Second, the logic $\mathbf{QHT}^s_=$ can be applied to characterise properties such as the strong equivalence of programs and theories [20, 27]. While strong equivalence and related concepts have been much studied recently in ASP, their characterisation in the case of hybrid KBs remains uncharted territory. The fact that QEL provides a single semantics for hybrid KBs means that a simple concept of strong equivalence is applicable to such KBs and characterisable using the underlying logic, $\mathbf{QHT}^s_=$. In Section 9 below we describe how $\mathbf{QHT}^s_=$ can be applied in this context.

8 Hybrid KBs and the SM operator

Recently, Ferraris, Lee and Lifschitz [12] have presented a new definition of stable models. It is applicable to sentences or finitely axiomatisable theories in first-order logic. The definition is syntactical and involves an operator SM that resembles parallel circumscription. The stable models of a sentence F are the structures that satisfy a certain second-order sentence, $\mathrm{SM}[F]$. This new definition of stable model agrees with equilibrium logic in the sense that the models of $\mathrm{SM}[F]$ from [12] are essentially the equilibrium models of F as defined in this article.

We shall now show that by slightly modifying the SM operator we can also capture the NM semantics of hybrid knowledge bases. First, we need to introduce some notation, essentially following [20].

If p and q are predicate constants of the same arity then $p = q$ stands for the formula

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow q(\mathbf{x})),$$

and $p \leq q$ stands for

$$\forall \mathbf{x}(p(\mathbf{x}) \rightarrow q(\mathbf{x})),$$

[11]For a recent extension of answer sets to first-order formulas, see [12], which is explained in more detail in Section 8.

where $\mathbf{x}$ is a tuple of distinct object variables. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ of predicate constants then $\mathbf{p} = \mathbf{q}$ stands for the conjunction

$$p_1 = q_1 \wedge \cdots \wedge p_n = q_n,$$

and $\mathbf{p} \leq \mathbf{q}$ for

$$p_1 \leq q_1 \wedge \cdots \wedge p_n \leq q_n.$$

Finally, $\mathbf{p} < \mathbf{q}$ is an abbreviation for $\mathbf{p} \leq \mathbf{q} \wedge \neg(\mathbf{p} = \mathbf{q})$. The operator $\mathrm{NM}|_{\mathcal{P}}$ defines second-order formulas and the previous notation can be also applied to tuples of predicate variables.

$$\mathrm{NM}|_{\mathcal{P}}[F] = F \wedge \neg\exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where $\mathbf{p}$ is the list of all predicate constants $p_1, \ldots, p_n \notin \mathcal{L}_{\mathcal{T}}$ occurring in F, $\mathbf{u}$ is a list of n distinct predicate variables $u_1, \ldots, u_n$. The $\mathrm{NM}|_{\mathcal{P}}$ operator works just like in the SM operator from [12] except that the substitution of predicates p_i is restricted to those not in $\mathcal{L}_{\mathcal{T}}$. Notice that in the definition of $\mathrm{NM}|_{\mathcal{P}}[F]$ the second conjunct specifies the minimality condition on interpretations while the third conjunct involves a translation '*' that provides a reduction of the non-classical here-and-there logic to classical logic. This translation is recursively defined as follows:

- $p_i(t_1, \ldots, t_m)^* = u_i(t_1, \ldots, t_m)$ if $p_i \notin \mathcal{L}_{\mathcal{T}}$;
- $p_i(t_1, \ldots, t_m)^* = p_i(t_1, \ldots, t_m)$ if $p_i \in \mathcal{L}_{\mathcal{T}}$;
- $(t_1 = t_2)^* = (t_1 = t_2)$;
- $\bot^* = \bot$;
- $(F \odot G)^* = F^* \odot G^*$, where $\odot \in \{\wedge, \vee\}$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(QxF)^* = QxF^*$, where $Q \in \{\forall, \exists\}$.

(There is no clause for negation here, because $\neg F$ is treated as shorthand for $F \to \bot$.)

Theorem 15 *$\mathcal{M} = \langle U, T\rangle$ is a NM-model of $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ if and only if it satisfies $\mathcal{T}$ and* $\mathrm{NM}|_{\mathcal{P}}[\mathcal{P}]$.

We assume here that both $\mathcal{T}$ and $\mathcal{P}$ are finite, so that the operator $\mathrm{NM}|_{\mathcal{P}}$ is well-defined.
Proof:

($\Rightarrow$) If $\langle U, T\rangle$, $U = (D, \sigma)$, is a NM-model of $\mathcal{K} = (\mathcal{T}, \mathcal{P})$, then $\langle U, T\rangle \models \mathcal{T}$, and $\langle U, T\rangle \models \mathcal{P}$, and $\langle U, T, T\rangle$ is an equilibrium model of $\mathcal{T} \cup st(\mathcal{T}) \cup \mathcal{P}$. So we only need to prove that $\langle U, T\rangle \models \neg\exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge P^*(\mathbf{u}))$. For the contradiction, let us assume that

$$\langle U, T\rangle \models \exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge \mathcal{P}^*(\mathbf{u}))$$

This means that:
Fact 1: For every $p_i \notin \mathcal{L}_\mathcal{T}$, there exists $\overline{p}_i \subset D^n$ such that $(\mathbf{u} < \mathbf{p}) \wedge \mathcal{P}^*(\mathbf{u})$ is valid in the structure $\langle U, T\rangle$ where u_i is interpreted as $\overline{p}_i$.

If we consider the set

$$H = \{p_i(d_1, \dots, d_k) \colon (d_1, \dots, d_k) \in \overline{p}_i\} \cup \\ \cup \{p_i(d_1, \dots, d_k) \colon p_i \in \mathcal{L}_\mathcal{T}, p_i(d_1, \dots, d_k) \in T\},$$

and u_i is interpreted as $\overline{p}_i$, then $\mathbf{u} < \mathbf{p}$ is valid in $\langle U, T\rangle$ iff $H \subset T$ and $\mathcal{P}^*(\mathbf{u})$ is valid in $\langle U, T\rangle$ iff $\langle U, H\rangle \models \mathcal{P}^*(\mathbf{p})$; that is, the *Fact 1* is equivalent to:

$$H \subset T \quad \text{and} \quad \langle U, H\rangle \models \mathcal{P}^*(\mathbf{p}) \tag{6}$$

Since $T \smallsetminus H$ does not include predicate symbols of $\mathcal{L}_\mathcal{T}$, $\langle U, H, T\rangle \models \mathcal{T} \cup st(\mathcal{T})$. So, to finish the proof, we need to prove the following for every formula φ:
Fact 2: $\langle U, H, T\rangle, h \models \varphi$ if and only if $\langle U, H\rangle \models \varphi^*(\mathbf{p})$.

As a consequence of *Fact 2*, we have that $\langle U, H, T\rangle \models \mathcal{P}$ and thus $\langle U, H, T\rangle$ is a model of the stable closure of $\mathcal{K}$, which contradicts that $\langle U, T, T\rangle$ is in equilibrium.

Fact 2 is proved by induction on φ:

(i) If $\varphi = p_i(d_1, \dots, d_k)$, then $\varphi^*(\mathbf{p}) = \varphi$:

$$\langle U, H, T\rangle, h \models \varphi \Leftrightarrow p_i(d_1, \dots, d_k) \in H \Leftrightarrow \langle U, H\rangle \models \varphi^*(\mathbf{p})$$

(ii) Let $\varphi = \psi_1 \wedge \psi_2$ and assume that, for $i = 1, 2$,

$$\langle U, H, T\rangle, h \models \psi_i \quad \text{iff} \quad \langle U, H\rangle \models {\psi_i}^*(\mathbf{p}). \tag{7}$$

$*$ For $\varphi = \psi_1 \wedge \psi_2$ under assumption (7):

$$\begin{aligned} \langle U, H, T\rangle, h \models \psi_1 \wedge \psi_2 &\Leftrightarrow \langle U, H, T\rangle, h \models \psi_1 \text{ and } \langle U, H, T\rangle, h \models \psi_2 \\ &\Leftrightarrow \langle U, H\rangle \models \psi_1^*(\mathbf{p}) \text{ and } \langle U, H\rangle \models \psi_2^*(\mathbf{p}) \\ &\Leftrightarrow \langle U, H\rangle \models (\psi_1 \wedge \psi_2)^* \end{aligned}$$

$*$ Similarly, for $\varphi = \psi_1 \to \psi_2$ under assumption (7):

$$\begin{aligned} &\langle U, H, T\rangle, h \models \psi_1 \to \psi_2 \Leftrightarrow \\ &\Leftrightarrow \langle U, H, T\rangle, t \models \psi_1 \to \psi_2 \text{ and} \\ &\qquad\qquad \text{either } \langle U, H, T\rangle, h \not\models \psi_1 \text{ or } \langle U, H, T\rangle, h \models \psi_2 \\ &\Leftrightarrow \langle U, T\rangle \models \psi_1 \to \psi_2 \text{ and either } \langle U, H\rangle \not\models \psi_1^*(\mathbf{p}) \text{ or } \langle U, H\rangle \models \psi_2^*(\mathbf{p}) \\ &\Leftrightarrow \langle U, H\rangle \models (\psi_1 \to \psi_2)^* \end{aligned}$$

$(\Leftarrow)$ If $\langle U, T\rangle$, $U = (D, \sigma)$, satisfies $\mathcal{T}$ and $\mathrm{NM}|_\mathcal{P}[\mathcal{P}]$, then trivially $\langle U, T, T\rangle$ is a here-and-there model of the closure of $\mathcal{K}$; we only need to prove that this model is in equilibrium. By contradiction, let us assume that $\langle U, H, T\rangle$ is a here-and-there model of the closure of $\mathcal{K}$ with $H \subset T$. For every $p_i \notin \mathcal{L}_\mathcal{T}$, we define

$$\overline{p}_i = \{(d_i, \ldots, d_k) \colon p_i(d_i, \ldots, d_k) \in H\}$$

Fact 3: $(\mathbf{u} < \mathbf{p}) \wedge \mathcal{P}^*(\mathbf{u})$ is valid in the structure $\langle U, T\rangle$ if the variables u_i are interpreted as $\overline{p}_i$.

As a consequence of *Fact 3*, we have that $\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge \mathcal{P}^*(\mathbf{u}))$ is satisfied by $\langle U, T\rangle$ which contradicts that $\mathrm{NM}|_\mathcal{P}[\mathcal{P}]$ is satisfied by the structure.

As in the previous item, *Fact 3* is equivalent to

$$H \subset T \quad \text{and} \quad \langle U, H\rangle \models \mathcal{P}^*(\mathbf{p})$$

The first condition, $H \subset T$, is trivial by definition and the second one is a consequence of *Fact 2*.

$\square$

9 The Strong Equivalence of Knowledge Bases

Let us see how the previous results, notably Theorem 13, can be applied to characterise a concept of strong equivalence between hybrid knowledge bases. It is important to know when different reconstructions of a given body of knowledge or state of affairs are equivalent and lead to essentially the same problem solutions. In the case of knowledge reconstructed in classical logic, ordinary logical equivalence can serve as a suitable concept when applied to theories formulated in the same vocabulary. In the case where nonmonotonic rules are present, however, the situation changes: two sets of rules may have the same answer sets yet behave very differently once they are embedded in some larger context. Thus for hybrid knowledge bases one may also like to know that equivalence is robust or modular. A robust notion of equivalence for logic programs will require that programs behave similarly when extended by any further programs. This leads to the following concept of *strong* equivalence: programs Π_1 and Π_2 are strongly equivalent if and only if for any set of rules Σ, $\Pi_1 \cup \Sigma$ and $\Pi_2 \cup \Sigma$ have the same answer sets. This concept of strong equivalence for logic programs in ASP was introduced and studied in [19] and has given rise to a substantial body of further work looking at different characterisations, new variations and applications of the idea, as well as the development of systems to test for strong equivalence. Strong equivalence has also been defined and studied for logic programs with variables and first-order nonmonotonic theories under the stable model or equilibrium logic semantics [22, 8, 20, 27]. In equilibrium logic we say that two (first-order) theories Π_1 and Π_2 are strongly equivalent if and only if for any theory Σ, $\Pi_1 \cup \Sigma$ and $\Pi_2 \cup \Sigma$ have the same equilibrium models [20, 27]. Under this definition we have:

Theorem 16 ([20, 27]) *Two (first-order) theories Π_1 and Π_2 are strongly equivalent if and only if they are equivalent in* $\mathbf{QHT}^s_=$.

Different proofs of Theorem 16 are given in [20] and [27]. For present purposes, the proof contained in [27] is more useful. It shows that if theories are not strongly equivalent, the set of formulas Σ such that $\Pi_1 \cup \Sigma$ and $\Pi_2 \cup \Sigma$ do not have the same equilibrium models can be chosen to have the form of implications $(A \rightarrow B)$ where A and B are atomic. So if we are interested in the case where Π_1 and Π_2 are sets of rules, Σ can also be regarded as a set of rules. We shall make use of this property below.

In the case of hybrid knowledge bases $\mathcal{K} = (\mathcal{T}, \mathcal{P})$, various kinds of equivalence can be specified, according to whether one or other or both of the components $\mathcal{T}$ and $\mathcal{P}$ are allowed to vary. The following form is rather general.

Definition 8 *Let $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{P}_2)$, be two hybrid KBs sharing the same structural language, ie. $\mathcal{L}_{\mathcal{T}1} = \mathcal{L}_{\mathcal{T}2}$. $\mathcal{K}_1$ and $\mathcal{K}_2$ are said to be* strongly equivalent *if for any theory $\mathcal{T}$ and set of rules $\mathcal{P}$, $(\mathcal{T}_1 \cup \mathcal{T}, \mathcal{P}_1 \cup \mathcal{P})$ and $(\mathcal{T}_2 \cup \mathcal{T}, \mathcal{P}_2 \cup \mathcal{P})$ have the same NM-models.*

Until further notice, let us suppose that $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ are hybrid KBs sharing a common structural language $\mathcal{L}$.

Proposition 17 *$\mathcal{K}_1$ and $\mathcal{K}_2$ are strongly equivalent if and only if $\mathcal{T}_1 \cup st(\mathcal{T}_1) \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup st(\mathcal{T}_2) \cup \mathcal{P}_2$ are logically equivalent in $\mathbf{QHT}^s_=$.*

Proof: Let $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ be hybrid KBs such that $\mathcal{L}_{\mathcal{T}1} = \mathcal{L}_{\mathcal{T}2} = \mathcal{L}$. Suppose $\mathcal{T}_1 \cup st(\mathcal{T}_1) \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup st(\mathcal{T}_2) \cup \mathcal{P}_2$ are logically equivalent in $\mathbf{QHT}^s_=$. Clearly $(\mathcal{T}_1 \cup \mathcal{T} \cup st(\mathcal{T}_1 \cup \mathcal{T}) \cup \mathcal{P}_1 \cup \mathcal{P})$ and $(\mathcal{T}_2 \cup \mathcal{T} \cup st(\mathcal{T}_2 \cup \mathcal{T}) \cup \mathcal{P}_2 \cup \mathcal{P})$ have the same $\mathbf{QHT}^s_=$-models and hence the same equilibrium models. Strong equivalence of $\mathcal{K}_1$ and $\mathcal{K}_2$ follows by Theorem 13.

For the 'only-if' direction, suppose that $\mathcal{T}_1 \cup st(\mathcal{T}_1) \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup st(\mathcal{T}_2) \cup \mathcal{P}_2$ are not logically equivalent in $\mathbf{QHT}^s_=$, so there is.an $\mathbf{QHT}^s_=$-model of one of these theories that is not an $\mathbf{QHT}^s_=$ of the other. Applying the proof of Theorem 16 given in [27] we can infer that there is a set $\mathcal{P}$ of rules of a simple sort such that the equilibrium models of $\mathcal{T}_1 \cup st(\mathcal{T}_1) \cup \mathcal{P}_1 \cup \mathcal{P}$ and $\mathcal{T}_2 \cup st(\mathcal{T}_2) \cup \mathcal{P}_2 \cup \mathcal{P}$ do not coincide. Hence by Theorem 13 $\mathcal{K}_1$ and $\mathcal{K}_2$ are not strongly equivalent. $\square$

Notice that from the proof of Proposition 17 it follows that the non-strong equivalence of two hybrid knowledge bases can always be made manifest by choosing extensions having a simple form, obtained by adding simple rules to the rule base.

We mention some conditions to test for strong equivalence and non-equivalence.

Corollary 18 *(a) $\mathcal{K}_1$ and $\mathcal{K}_2$ are strongly equivalent if $\mathcal{T}_1$ and $\mathcal{T}_2$ are classically equivalent and $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent in $\mathbf{QHT}^s_=$.*
(b) $\mathcal{K}_1$ and $\mathcal{K}_2$ are not strongly equivalent if $\mathcal{T}_1 \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup \mathcal{P}_2$ are not equivalent in classical logic.

Proof:
(a) Assume the hypothesis. Since $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ share a common structural language $\mathcal{L}$, it follows that $st(\mathcal{T}_1) = st(\mathcal{T}_2) = \mathcal{S}$, say. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are classically equivalent, $\mathcal{T}_1 \cup \mathcal{S}$ and $\mathcal{T}_2 \cup \mathcal{S}$ have the same (total) $\mathbf{QHT}^s_=$-models and so for any $\mathcal{T}$ also $\mathcal{T}_1 \cup \mathcal{T} \cup \mathcal{S} \cup st(\mathcal{T})$ and $\mathcal{T}_2 \cup \mathcal{T} \cup \mathcal{S} \cup st(\mathcal{T})$ have the same (total)

$\mathbf{QHT}^s_=$-models. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent in $\mathbf{QHT}^s_=$ it follows also that for any $\mathcal{P}$, $(\mathcal{T}_1 \cup \mathcal{T} \cup \mathcal{S} \cup st(\mathcal{T}) \cup \mathcal{P}_1 \cup \mathcal{P})$ and $(\mathcal{T}_2 \cup \mathcal{T} \cup \mathcal{S} \cup st(\mathcal{T}) \cup \mathcal{P}_2 \cup \mathcal{P})$ have the same $\mathbf{QHT}^s_=$-models and hence the same equilibrium models. The conclusion follows by Theorem 13.

(b) Suppose that $\mathcal{T}_1 \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup \mathcal{P}_2$ are not equivalent in classical logic. Assume again that $st(\mathcal{T}_1) = st(\mathcal{T}_2) = \mathcal{S}$, say. Then clearly $\mathcal{T}_1 \cup \mathcal{S} \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup \mathcal{S} \cup \mathcal{P}_2$ are not classically equivalent and hence they cannot be $\mathbf{QHT}^s_=$-equivalent. Applying the second part of the proof of Proposition 17 completes the argument. □

Special cases of strong equivalence arise when hybrid KBs are based on the same classical theory, say, or share the same rule base. That is, $(\mathcal{T}, \mathcal{P}_1)$ and $(\mathcal{T}, \mathcal{P}_2)$ are strongly equivalent if $\mathcal{P}_1$ and $\mathcal{P}_2$ are $\mathbf{QHT}^s_=$-equivalent.[12] Analogously:

$$(\mathcal{T}_1, \mathcal{P}) \text{ and } (\mathcal{T}_2, \mathcal{P}) \text{ are strongly equivalent if } \mathcal{T}_1 \text{ and } \mathcal{T}_2 \text{ are classically equivalent.} \quad (8)$$

Let us briefly comment on a restriction that we imposed on strong equivalence, namely that the KBs in question share a common structural language. Intuitively the reason for this is that the structural language $\mathcal{L}_\mathcal{T}$ associated with a hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ is part of its identity or 'meaning'. Precisely the predicates in $\mathcal{L}_\mathcal{T}$ are the ones treated classically. In fact, another KB, $\mathcal{K}' = (\mathcal{T}', \mathcal{P})$, where $\mathcal{T}'$ is completely equivalent to $\mathcal{T}$ in classical logic, may have a different semantics if $\mathcal{L}_\mathcal{T}'$ is different from $\mathcal{L}_\mathcal{T}$. To see this, let us consider a simple example in propositional logic. Let $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{P}_2)$, be two hybrid KBs where $\mathcal{P}_1 = \mathcal{P}_2 = \{(p \rightarrow q)\}$, $\mathcal{T}_1 = \{(r \wedge (r \vee p))\}$, $\mathcal{T}_2 = \{r\}$. Clearly, $\mathcal{T}_1$ and $\mathcal{T}_2$ are classically and even $\mathbf{QHT}^s_=$-equivalent. However, $\mathcal{K}_1$ and $\mathcal{K}_2$ are not even in a weak sense semantically equivalent. $st(\mathcal{T}_1) = \{r \vee \neg r; p \vee \neg p\}$, while $st(\mathcal{T}_2) = \{r \vee \neg r\}$. It is easy to check that $\mathcal{T}_1 \cup st(\mathcal{T}_1) \cup \mathcal{P}_1$ and $\mathcal{T}_2 \cup st(\mathcal{T}_2) \cup \mathcal{P}_2$ have different $\mathbf{QHT}^s_=$-models, different equilibrium models and (hence) $\mathcal{K}_1$ and $\mathcal{K}_2$ have different NM-models. So we see that without the assumption of a common structural language, the natural properties expressed in Corollary 18 (a) and (8) would no longer hold.

It is interesting to note here that meaning-preserving relations among ontologies have recently become a topic of interest in the description logic community where logical concepts such as that of conservative extension are currently being studied and applied [13]. A unified, logical approach to hybrid KBs such as that developed here should lend itself well to the application of such concepts.

10 Related Work and Conclusions

We have provided a general notion of hybrid knowledge base, combining first-order theories with nonmonotonic rules, with the aim of comparing and contrasting some of the different variants of hybrid KBs found in the literature [29, 30, 31, 16]. We presented a version of quantified equilibrium logic, QEL, without the unique names assumption, as a unified logical foundation for hybrid knowledge bases. We showed how for a hybrid knowledge base $\mathcal{K}$ there is a natural correspondence between the

[12]In [5] it was incorrectly stated in Proposition 7 that this condition was both necessary and sufficient for strong equivalence, instead of merely sufficient.

nonmonotonic models of $\mathcal{K}$ and the equilibrium models of what we call the *stable closure* of $\mathcal{K}$. This yields a way to capture in QEL the semantics of the g-hybrid KBs of Heymans et al. [16] and the r-hybrid KBs of Rosati [30], where the latter is defined without the UNA but for safe programs. Similarly, the version of QEL with UNA captures the semantics of r-hybrid KBs as defined in [29, 31]. It is important to note that the aim of this paper was not that of providing new kinds of safety conditions or decidability results; these issues are ably dealt with in the literature reviewed here. Rather our objective has been to show how classical and nonmonotonic theories might be unified under a single semantical model. In part, as [16] show with their reduction of DL knowledge bases to open answer set programs, this can also be achieved (at some cost of translation) in other approaches. What distinguishes QEL is the fact that it is based on a standard, nonclassical logic, $\mathbf{QHT}^s_=$, which can therefore provide a unified logical foundation for such extensions of (open) ASP. To illustrate the usefulness of our framework we showed how the logic $\mathbf{QHT}^s_=$ also captures a natural concept of strong equivalence between hybrid knowledge bases.

There are several other approaches to combining languages for Ontologies with nonmonotonic rules which can be divided into two main streams [3]: approaches which define integration of rules and ontologies (a) by entailment, ie. querying classical knowledge bases through special predicates the rules body, and (b) on the basis of single models, ie. defining a common notion of combined model.

The most prominent of the former kind of approaches are dl-programs [10] and their generalization, HEX-programs [9]. Although these approaches both are based on Answer Set programming like our approach, the orthogonal view of integration by entailment can probably not be captured by a simple embedding in QEL. Another such approach which allows querying classical KBs from a nonmonotonic rules language is based on Defeasible Logic [33].

As for the second stream, variants of Autoepistemic Logic [4], and the logic of minimal knowledge and negation as failure (MKNF) [23] have been recently proposed in the literature. Similar to our approach, both these approaches embed a combined knowledge base in a unifying logic. However, both purchase use modal logics fact syntactically and semantically extend first-order logics. Thus, in these approaches, embedding of the classical part of the theory is trivial, whereas the nonmonotonic rules part needs to be rewritten in terms of modal formulas. Our approach is orthogonal, as we use a non-classical logic where the nonmonotonic rules are trivially embedded, but the stable closure guarantees classical behavior of certain predicates. In addition, the fact that we include the stable closure ensures that the predicates from the classical parts of the theory behave classically, also when used in rules with negation. In contrast, in both modal approaches occurrences of classical predicates are not interpreted classically, as illustrated in the following example.

Example 6 *Consider the theory $\mathcal{T} = \{A(a)\}$ and the program $\mathcal{P} = \{r \leftarrow \neg A(b)\}$. We have that there exists an NM-model $\mathcal{M}$ of $(\mathcal{T}, \mathcal{P})$ such that $\mathcal{M} \models A(a), A(b)$ and $\mathcal{M} \not\models A(a), A(b)$, and so r is not entailed. Consider now the embedding τ_{HP} of logic programs into autoepistemic logic [4]. We have $\tau_{HP}(\mathcal{P}) = \{\neg \mathsf{L}A(b) \rightarrow r\}$. In autoepistemic logic, $\mathsf{L}A(b)$ is true iff $A(b)$ is included in a* stable expansion *T, which is essentially the set of all entailed formulas, assuming T. We have that $A(b)$ is not*

entailed from $\mathcal{T} \cup \tau_{HP}(\mathcal{P})$ under any stable expansion, and so $\mathsf{L}A(b)$ is false, and thus r is necessarily true in every model. We thus have that r is a consequence of $\mathcal{T} \cup \tau_{HP}(\mathcal{P})$.
Similar for the hybrid MKNF knowledge bases by [23].

As shown by [6], adding *classical interpretation axioms* – essentially a modal version of the stable closure axioms – to the theory $\mathcal{T} \cup \tau_{HP}(\mathcal{P})$ allows one to capture the hybrid knowledge base semantics we considered in this paper.

In future work we hope to consider further aspects of applying QEL to the domain of hybrid knowledge systems. Extending the language with functions symbols and with strong negation is a routine task, since QEL includes these items already. We also plan to consider in the future how QEL can be used to define a catalogue of logical relations between hybrid KBs. Last, but not least, let us mention that in this paper we exclusively dealt with hybrid combinations of classical theories with logic programs under variants of the stable-model semantics. Recenty, also hybrid rule combinations based on the well-founded semantics have been proposed by Drabent et al. [7] or Knorr et al. [18], defining an analogous, modular semantics like hybrid knowledge bases considered here. In this context, we plan to investigate whether a first-order version of Partial Equilibrium Logic [2], which has been recently shown to capure the well-founded semantics in the propositional case, can simlarly work as a foundation for hybrid rule combintions à la Drabent et al. [7].

We believe that on the long run the general problem addressed by the theoretical foundations layed in this paper could potentially provide essential insights for realistic applications of Ontologies and Semantic Web technologies in general, since for most of these applications current classical ontology languages provide too limited expresivity and the addition of non-monotonic rules is key to overcome these limitations. As an example let us mention the "clash" between the open world assumption in Ontologies and the nonmonotonicity/closed world nature of typical Semantic Web query languages such as SPARQL, which contains nonmonotonic constructs and in fact can be translated to rules with nonmonotonic negation [28]. While some initial works exist in the direction of using SPARQL on top of OWL [17], the foundations and exact semantic treatment of corner cases is still an open problem.[13] As another example, let us mention mappings between modular ontologies as for instance investigated by [11]; non-monotonic rules could provide a powerful tool to describe mappings between ontologies.

Acknowledgements

Part of the results in this paper are contained, in preliminary form, in the proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR2007), and in the informal proceedings of the RuleML-06 Workshop on Ontology and Rule Integration. The authors thank the anonymous reviewers of those preliminary versions of the article for their helpful comments. This research has been partially supported

[13]One of the authors of the present paper is in fact chairing the W3C SPARQL working group in which at the time of writing of this paper this topic has been being discussed actively.

by the Spanish MEC (now MCI) under the projects TIC-2003-9001, TIN2006-15455-CO3, and CSD2007-00022, also by the project URJC-CM-2006-CET-0300 and by the European Commission under the projects Knowledge Web (IST-2004-507482) and inContext (IST-034718), as well as by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

[1] Baral, C. (2002), *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press.

[2] Cabalar, P., Odintsov, S. P., Pearce, D., and Valverde, A. (2006), Analysing and extending well-founded and partial stable semantics using partial equilibrium logic, *in* 'Proceedings of the 22nd International Conference on Logic Programming (ICLP 2006)', Vol. 4079 of *Lecture Notes in Computer Science*, Springer, Seattle, WA, USA, pp. 346–360.

[3] de Bruijn, J., Eiter, T., Polleres, A., and Tompits, H. (2006), On representational issues about combinations of classical theories with nonmonotonic rules, *in* 'Proceedings of the First International Conference on Knowledge Science, Engineering and Management (KSEM'06)', number 4092 *in* 'Lecture Notes in Computer Science', Springer-Verlag, Guilin, China.

[4] de Bruijn, J., Eiter, T., Polleres, A., and Tompits, H. (2007), Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination, *in* 'Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)', AAAI, Hyderabad, India, pp. 304–309.

[5] de Bruijn, J., Pearce, D., Polleres, A., and Valverde, A. (2007), Quantified equilibrium logic and hybrid rules, *in* 'First International Conference on Web Reasoning and Rule Systems (RR2007)', Vol. 4524 of *Lecture Notes in Computer Science*, Springer, Innsbruck, Austria, pp. 58–72.

[6] de Bruijn, J., Eiter, T., and Tompits, H. (2008), Embedding approaches to combining rules and ontologies into autoepistemic logic, *in* 'Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR2008)', AAAI, Sydney, Australia, pp. 485–495.

[7] Drabent, W., Henriksson, J., and Maluszynski, J. (2007), Hybrid reasoning with rules and constraints under well-founded semantics, *in* 'First International Conference on Web Reasoning and Rule Systems (RR2007)', Vol. 4524 of *Lecture Notes in Computer Science*, Springer, Innsbruck, Austria, pp. 348–357.

[8] Eiter, T., Fink, M., Tompits, H., and Woltran, S. (2005), Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case, *in* 'Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference', pp. 695–700.

[9] Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2005), A uniform integration of higher-order reasoning and external evaluations in answer-set programming, *in* 'IJCAI 2005', pp. 90–96.

[10] Eiter, T., Lukasiewicz, T., Schindlauer, R., and Tompits, H. (2004), Combining answer set programming with description logics for the semantic Web, *in* 'Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)'.

[11] Ensan, F. and Du, W. (2009), A knowledge encapsulation approach to ontology modularization. *Knowledge and Information Systems* **Online First**, to appear.

[12] Ferraris, P., Lee, J., and Lifschitz, V. (2007), A new perspective on stable models, *in* 'Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)', AAAI, Hyderabad, India, pp. 372–379.

[13] Ghilardi, S., Lutz, C., and Wolter, F. (2006), Did I damage my ontology: A Case for Conservative Extensions of Description Logics, *in* 'Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)', pp. 187–197.

[14] Heymans, S. (2006), Decidable Open Answer Set Programming, PhD thesis, Theoretical Computer Science Lab (TINF), Department of Computer Science, Vrije Universiteit Brussel, Brussels, Belgium.

[15] Heymans, S., Nieuwenborgh, D. V., and Vermeir, D. (2005), Guarded Open Answer Set Programming, *in* '8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005)', volume 3662 *in* 'LNAI', Springer, pp. 92–104.

[16] Heymans, S., Predoiu, L., Feier, C., de Bruijn, J., and van Nieuwenborgh, D. (2006), G-hybrid knowledge bases, *in* 'Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)'.

[17] Jing, Y., Jeong, D., and Baik, D.-K. (2009), SPARQL graph pattern rewriting for OWL-DL inference queries. *Knowledge and Information Systems* **20**, pp. 243–262.

[18] Knorr, M., Alferes, J., and Hitzler, P. (2008), A coherent well-founded model for hybrid MKNF knowledge bases, *in* '18th European Conference on Artificial Intelligence (ECAI2008)', volume 178 *in* 'Frontiers in Artificial Intelligence and Applications', IOS Press, pp. 99–103.

[19] Lifschitz, V., Pearce, D., and Valverde, A. (2001), 'Strongly equivalent logic programs', *ACM Transactions on Computational Logic* **2**(4), 526–541.

[20] Lifschitz, V., Pearce, D., and Valverde, A. (2007), A characterization of strong equivalence for logic programs with variables, *in* '9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR))', Vol. 4483 of *Lecture Notes in Computer Science*, Springer, Tempe, AZ, USA, pp. 188–200.

[21] Lifschitz, V. and Woo, T. (1992), Answer sets in general nonmonotonic reasoning (preliminary report), *in* B. Nebel, C. Rich and W. Swartout, eds, 'KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference', Morgan Kaufmann, San Mateo, California, pp. 603–614.

[22] Lin, F. (2002), Reducing strong equivalence of logic programs to entailment in classical propositional logic, *in* 'Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning (KR'02)', pp. 170–176.

[23] Motik, B. and Rosati, R. (2007), A faithful integration of description logics with logic programming, *in* 'Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)', AAAI, Hyderabad, India, pp. 477–482.

[24] Pearce, D. (1997), A new logical characterization of stable models and answer sets, *in* 'Proceedings of NMELP 96', Vol. 1216 of *Lecture Notes in Computer Science*, Springer, pp. 57–70.

[25] Pearce, D. (2006), 'Equilibrium logic', *Annals of Mathematics and Artificial Intelligence* **47** 3–41.

[26] Pearce, D. and Valverde, A. (2005), 'A first-order nonmonotonic extension of constructive logic', *Studia Logica* **80**, 321–246.

[27] Pearce, D. and Valverde, A. (2006), Quantfied equilibrium logic, Technical report, Universidad Rey Juan Carlos. in press.

[28] Polleres, A. (2007), From SPARQL to Rules (and back), *in* 'WWW 2007', pp. 787–796.

[29] Rosati, R. (2005*a*), 'On the decidability and complexity of integrating ontologies and rules', *Journal of Web Semantics* **3**(1), 61–73.

[30] Rosati, R. (2005*b*), Semantic and computational advantages of the safe integration of ontologies and rules, *in* 'Proceedings of the Third International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2005)', Vol. 3703 of *Lecture Notes in Computer Science*, Springer, pp. 50–64.

[31] Rosati, R. (2006), $\mathcal{DL} + log$: Tight integration of description logics and disjunctive datalog, *in* 'Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)', pp. 68–78.

[32] van Dalen, D. (1983), *Logic and Structure*, Springer.

[33] Wang, K., Billington, D., Blee, J., and Antoniou, G. (2004), Combining description logic and defeasible logic for the semantic Web, *in* 'Proceedings of the Third International Workshop Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)', Vol. 3323 of *Lecture Notes in Computer Science*, Springer, pp. 170–181.

Published in Proceedings of the 16th World Wide Web Conference (WWW2007), pp. 787–496,
May 2007, ACM Press

From SPARQL to Rules (and back) [*]

Axel Polleres
Digital Enterprise Research Institute, National University of Ireland, Galway
axel@polleres.net

Abstract

As the data and ontology layers of the Semantic Web stack have achieved a certain level of maturity in standard recommendations such as RDF and OWL, the current focus lies on two related aspects. On the one hand, the definition of a suitable query language for RDF, SPARQL, is close to recommendation status within the W3C. The establishment of the rules layer on top of the existing stack on the other hand marks the next step to be taken, where languages with their roots in Logic Programming and Deductive Databases are receiving considerable attention. The purpose of this paper is threefold. First, we discuss the formal semantics of SPARQL extending recent results in several ways. Second, we provide translations from SPARQL to Datalog with negation as failure. Third, we propose some useful and easy to implement extensions of SPARQL, based on this translation. As it turns out, the combination serves for direct implementations of SPARQL on top of existing rules engines as well as a basis for more general rules and query languages on top of RDF.

Categories and Subject Descriptors: H.2.3[Languages]: Query Languages; H.3.5[Online Information Services]: Web-based services

General Terms: Languages, Standardization

Keywords: SPARQL, Datalog, Rules

1 Introduction

After the data and ontology layers of the Semantic Web stack have achieved a certain level of maturity in standard recommendations such as RDF and OWL, the query and the rules layers seem to be the next building-blocks to be finalized. For the first part, SPARQL [18], W3C's proposed query language, seems to be close to recommendation, though the Data Access working group is still struggling with defining aspects such as a formal semantics or layering on top of OWL and RDFS. As for the second

[*]An extended technical report version of this article is available at `http://www.polleres.net/TRs/GIA-TR-2006-11-28.pdf`. This work was mainly conducted under a Spanish MEC grant at Universidad Rey Juan Carlos, Móstoles, Spain.

part, the RIF working group [1], who is responsible for the rules layer, is just producing first concrete results. Besides aspects like business rules exchange or reactive rules, deductive rules languages on top of RDF and OWL are of special interest to the RIF group. One such deductive rules language is Datalog, which has been successfully applied in areas such as deductive databases and thus might be viewed as a query language itself. Let us briefly recap our starting points:

Datalog and SQL. Analogies between Datalog and relational query languages such as SQL are well-known and -studied. Both formalisms cover UCQ (unions of conjunctive queries), where Datalog adds recursion, particularly unrestricted recursion involving nonmonotonic negation (aka unstratified negation as failure). Still, SQL is often viewed to be more powerful in several respects. On the one hand, the lack of recursion has been partly solved in the standard's 1999 version [20]. On the other hand, aggregates or external function calls are missing in pure Datalog. However, also developments on the Datalog side are evolving and with recent extensions of Datalog towards Answer Set Programming (ASP) – a logic programming paradigm extending and building on top of Datalog – lots of these issues have been solved, for instance by defining a declarative semantics for aggregates [9], external predicates [8].

The Semantic Web rules layer. Remarkably, logic programming dialects such as Datalog with nonmonotonic negation which are covered by Answer Set Programming are often viewed as a natural basis for the Semantic Web rules layer [7]. Current ASP systems offer extensions for retrieving RDF data and querying OWL knowledge bases from the Web [8]. Particular concerns in the Semantic Web community exist with respect to adding rules including nonmonotonic negation [3] which involve a form of closed world reasoning on top of RDF and OWL which both adopt an open world assumption. Recent proposals for solving this issue suggest a "safe" use of negation as failure over finite contexts only for the Web, also called *scoped negation* [17].

The Semantic Web query layer – SPARQL. Since we base our considerations in this paper on the assumption that similar correspondences as between SQL and Datalog can be established for SPARQL, we have to observe that SPARQL inherits a lot from SQL, but there also remain substantial differences: On the one hand, SPARQL does not deal with nested queries or recursion, a detail which is indeed surprising by the fact that SPARQL is a graph query language on RDF where, typical recursive queries such as transitive closure of a property might seem very useful. Likewise, aggregation (such as count, average, etc.) of object values in RDF triples which might appear useful have not yet been included in the current standard. On the other hand, subtleties like blank nodes (aka bNodes), or optional graph patterns, which are similar but (as we will see) different to outer joins in SQL or relational algebra, are not straightforwardly translatable to Datalog.

The goal of this paper is to shed light on the actual relation between declarative rules languages such as Datalog and SPARQL, and by this also provide valuable input for the currently ongoing discussions on the Semantic Web rules layer, in particular its integration with SPARQL, taking the likely direction into account that LP style rules languages will play a significant role in this context.

[1]`http://www.w3.org/2005/rules/wg`

Although the SPARQL specification does not seem 100% stable at the current point, just having taken a step back from candidate recommendation to working draft, we think that it is not too early for this exercise, as we will gain valuable insights and positive side effects by our investigation. More precisely, the contributions of the present work are:

- We refine and extend a recent proposal to formalize the semantics of SPARQL from Pérez et al. [16], presenting three variants, namely c-joining, s-joining and b-joining semantics where the latter coincides with [16], and can thus be considered normative. We further discuss how aspects such compositionality, or idempotency of joins are treated in these semantics.
- Based on the three semantic variants, we provide translations from a large fragment of SPARQL queries to Datalog, which give rise to implementations of SPARQL on top of existing engines.
- We provide some straightforward extensions of SPARQL such as a set difference operator MINUS, and nesting of ASK queries in FILTER expressions.
- Finally, we discuss an extension towards recursion by allowing bNode-free-CONSTRUCT queries as part of the query dataset, which may be viewed as a light-weight, recursive rule language on top of of RDF.

The remainder of this paper is structured as follows: In Sec. 2 we first overview SPARQL, discuss some issues in the language (Sec. 2.1) and then define its formal semantics (Sec. 2.2). After introducing a general form of Datalog with negation as failure under the answer set semantics in Sec. 3, we proceed with the translations of SPARQL to Datalog in Sec. 4. We finally discuss the above-mentioned language extensions in Sec. 5, before we conclude in Sec. 6.

2 RDF and SPARQL

In examples, we will subsequently refer to the two RDF graphs in Fig. 1 which give some information about Bob and $Alice$. Such information is common in FOAF files which are gaining popularity to describe personal data. Similarities with existing examples in [18] are on purpose. We assume the two RDF graphs given in TURTLE [2] notation and accessible via the IRIs `ex.org/bob` and `alice.org`[2]

We assume the pairwise disjoint, infinite sets I, B, L and Var, which denote IRIs, Blank nodes, RDF literals, and variables respectively. In this paper, an *RDF Graph* is then a finite set, of triples from $I \cup B \cup L \times I \times I \cup B \cup L$,[3] dereferenceable by an IRI. A SPARQL *query* is a quadruple $Q = (V, P, DS, SM)$, where V is a result form, P is a graph pattern, DS is a dataset, and SM is a set of solution modifiers. We refer to [18] for syntactical details and will explain these in the following as far as necessary. In this

[2]For reasons of legibility and conciseness, we omit the leading 'http://' or other schema identifiers in IRIs.

[3]Following SPARQL, we are slightly more general than the original RDF specification in that we allow literals in subject positions.

```
# Graph: ex.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix bob: <ex.org/bob#> .

  <ex.org/bob> foaf:maker _:a.
  _:a a foaf:Person ; foaf:name "Bob";
           foaf:knows _:b.

  _:b a foaf:Person ; foaf:nick "Alice".
  <alice.org/> foaf:maker _:b
```

```
# Graph: alice.org

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix alice: <alice.org#> .

  alice:me a foaf:Person ; foaf:name "Alice" ;
           foaf:knows _:c.

  _:c  a foaf:Person ; foaf:name "Bob" ;
            foaf:nick "Bobby".
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT?Y ?X
FROM <alice.org>
FROM <ex.org/bob>
WHERE { ?Y foaf:name ?X .}
```

?X	?Y
"Bob"	_:a
"Bob"	_:c
"Alice"	alice.org#me

Figure 1: Two RDF graphs in TURTLE notation and a simple SPARQL query.

paper, we will ignore solution modifiers mostly, thus we will usually write queries as triples $Q = (V, P, DS)$, and will use the syntax for graph patterns introduced below.

Result Forms Since we will, to a large extent, restrict ourselves to SELECT queries, it is sufficient for our purposes to describe result forms by sets variables. Other result forms will be discussed in Sec. 5. For instance, let $Q = (V, P, DS)$ denote the query from Fig. 1, then $V = \{?X, ?Y\}$. Query results in SPARQL are given by partial, i.e. possibly incomplete, substitutions of variables in V by RDF terms. In traditional relational query languages, such incompleteness is usually expressed using null values. Using such null values we will write solutions as tuples where the order of columns is determined by *lexicographically ordering* the variables in V. Given a set of variables V, let $\overline{V}$ denote the tuple obtained from lexicographically ordering V.

The query from Fig. 1 with result form $\overline{V} = (?X, ?Y)$ then has solution tuples ("Bob", _:a), ("Alice", alice.org#me), ("Bob", _:c). We write substitutions in sqare brackets, so these tuples correspond to the substitutions $[?X \rightarrow$ "Bob", $?Y \rightarrow$ _ : a], $[?X \rightarrow$ "Alice", $?Y \rightarrow$ alice.org#me], and $[?X \rightarrow$ "Bob", $?Y \rightarrow$ _ : c], respectively.

Graph Patterns We follow the recursive definition of graph patterns P from [16]:

- a tuple (s, p, o) is a graph pattern where $s, o \in I \cup L \cup Var$ and $p \in I \cup Var$.[4]
- if P and P' are graph patterns then $(P$ AND $P')$, $(P$ OPT $P')$, $(P$ UNION $P')$, $(P$ MINUS $P')$ are graph patterns.[5]

[4]We do not consider bNodes in patterns as these can be semantically equivalently replaced by variables in graph patterns [6].

[5]Note that AND and MINUS are not designated keywords in SPARQL, but we use them here for reasons of readability and in order to keep with the operator style definition of [16]. MINUS is syntactically not present at all, but we will suggest a syntax extension for this particular keyword in Sec. 5.

- if P is a graph pattern and $i \in I \cup Var$, then $(\textsf{GRAPH}\ i\ P)$ is a graph pattern.
- if P is a graph pattern and R is a filter expression then $(P\ \textsf{FILTER}\ R)$ is a graph pattern.

For any pattern P, we denote by $vars(P)$ the set of all variables occurring in P. As *atomic filter expression*, SPARQL allows the unary predicates BOUND, isBLANK, isIRI, isLITERAL, binary equality predicates '$=$' for literals, and other features such as comparison operators, data type conversion and string functions which we omit here, see [18, Sec. 11.3] for details. *Complex filter expressions* can be built using the connectives '$\neg$','$\wedge$','$\vee$'.

Datasets The dataset $DS = (G, \{(g_1, G_1), \dots (g_k, G_k)\})$ of a SPARQL query is defined by a default graph G plus a set of named graphs, i.e. pairs of IRIs and corresponding graphs. Without loss of generality (there are other ways to define the dataset such as in a SPARQL protocol query), we assume G given as the merge of the graphs denoted by the IRIs given in a set of FROM and FROM NAMED clauses. For instance, the query from Fig. 1 refers to the dataset which consists of the default graph obtained from merging `alice.org` $\uplus$ `ex.org/bob` plus an empty set of named graphs.

The relation between names and graphs in SPARQL is defined solely in terms of that the IRI defines a resource which is represented by the respective graph. In this paper, we assume that the IRIs represent indeed network-accessible resources where the respective RDF-graphs can be retrieved from. This view has also be taken e.g. in [17]. Particularly, this treatment is not to be confused with so-called named graphs in the sense of [4]. We thus identify each IRI with the RDF graph available at this IRI and each set of IRIs with the graph merge [13] of the respective IRIs. This allows us to identify the dataset by a pair of sets of IRIs $DS = (G, G_n)$ with $G = \{d_1, \dots, d_n\}$ and $G_n = \{g_1, \dots, g_k\}$ denoting the (merged) default graph and the set of named graphs, respectively. Hence, the following set of clauses

```
FROM <ex.org/bob>
FROM NAMED <alice.org>
```

defines the dataset $DS = (\{\texttt{ex.org/bob}\}, \{\texttt{alice.org}\})$.

2.1 Assumptions and Issues

In this section we will discuss some important issues about the current specification, and how we will deal with them here.

First, note that the default graph if specified by name in a FROM clause is not counted among the named graphs automatically [18, section 8, definition 1]. An unbound variable in the GRAPH directive, means any of the named graphs in DS, but does NOT necessarily include the default graph.

Example 7 *This issue becomes obvious in the following query with dataset* $DS = (\{\texttt{ex.org/bob}\}, \emptyset)$ *which has an empty solution set.*

```
SELECT ?N WHERE {?G foaf:maker ?M .
            GRAPH ?G { ?X foaf:name ?N } }
```

We will sometimes find the following assumption convenient to avoid such arguably unintuitive effects:

Definition 9 *(*Dataset closedness assumption*) Given a dataset $DS = (G, G_n)$, G_n implicitly contains (i) all graphs mentioned in G and (ii) all IRIs mentioned explicitly in the graphs corresponding to G.*

Under this assumption, the previous query has both $(''Alice'')$ and $(''Bob'')$ in its solution set.

Some more remarks are in place concerning FILTER expressions. According to the SPARQL specification *"Graph pattern matching creates bindings of variables [where] it is possible to further restrict solutions by constraining the allowable bindings of variables to RDF Terms [with FILTER expressions]."* However, it is not clearly specified how to deal with filter constraints referring to variables which do not appear in simple graph patterns. In this paper, for graph patterns of the form $(P \text{ FILTER } R)$ we tacitly assume *safe filter expressions*, i.e. that all variables used in a filter expression R also appear in the corresponding pattern P. This corresponds with the notion of safety in Datalog (see Sec.3), where the built-in predicates (which obviously correspond to filter predicates) do not suffice to safe unbound variables.

Moreover, the specification defines errors to avoid mistyped comparisons, or evaluation of built-in functions over unbound values, i.e. "*any potential solution that causes an error condition in a constraint will not form part of the final results, but does not cause the query to fail.*" These errors propagate over the whole FILTER expression, also over negation, as shown by the following example.

Example 8 *Assuming the dataset does not contain triples for the* `foaf : dummy` *property, the example query*

```
SELECT ?X
WHERE { {?X a foaf:Person .
          OPTIONAL { ?X foaf:dummy ?Y . } }
        FILTER ( ¬(isLITERAL (?Y)) ) }
```

would discard any solution for ?X*, since the unbound value for* ?Y *causes an error in the isLITERAL expression and thus the whole FILTER expression returns an error.*

We will take special care for these errors, when defining the semantics of FILTER expressions later on.

2.2 Formal Semantics of SPARQL

The semantics of SPARQL is still not formally defined in its current version. This lack of formal semantics has been tackled by a recent proposal of Pérez et al. [16]. We will base on this proposal, but suggest three variants thereof, namely (a) *bravely joining*, (b) *cautiously-joining*, and (c) *strictly-joining* semantics. Particularly, our definitions vary from [16] in the way we define joining unbound variables. Moreover, we will refine their notion of FILTER satisfaction in order to deal with error propagation properly.

We denote by T_{null} the union $I \cup B \cup L \cup \{\mathsf{null}\}$, where null is a dedicated constant denoting the unknown value not appearing in any of I, B, or L, how it is commonly introduced when defining outer joins in relational algebra.

A *substitution* θ from Var to T_{null} is a partial function $\theta : Var \rightarrow T_{\mathsf{null}}$. We write substitutions in postfix notation: For a triple pattern $t = (s, p, o)$ we denote by $t\theta$ the triple $(s\theta, p\theta, o\theta)$ obtained by applying the substitution to all variables in t. The *domain* of θ, $dom(\theta)$, is the subset of Var where θ is defined. For a substitution θ and a set of variables $D \subseteq Var$ we define the substitution θ^D with domain D as follows:

$$x\theta^D = \begin{cases} x\theta \text{ if } x \in dom(\theta) \cap D \\ \mathsf{null} \text{ if } x \in D \setminus dom(\theta) \end{cases}$$

Let θ_1 and θ_2 be substitutions, then $\theta_1 \cup \theta_2$ is the substitution obtained as follows:

$$x(\theta_1 \cup \theta_2) = \begin{cases} x\theta_1 \text{ if } x\theta_1 \text{ defined and } x\theta_2 \text{ undefined} \\ \text{else: } x\theta_1 \text{ if } x\theta_1 \text{ defined and } x\theta_2 = \mathsf{null} \\ \text{else: } x\theta_2 \text{ if } x\theta_2 \text{ defined} \\ \text{else: undefined} \end{cases}$$

Thus, in the union of two substitutions defined values in one take precedence over null values the other substitution. For instance, given the substitutions $\theta_1 = [?X \rightarrow "Alice", ?Y \rightarrow \_:\mathsf{a}, ?Z \rightarrow \mathsf{null}]$ and $\theta_2 = [?U \rightarrow "Bob", ?X \rightarrow "Alice", ?Y \rightarrow \mathsf{null}]$ we get: $\theta_1 \cup \theta_2 = [?U \rightarrow "Bob", ?X \rightarrow "Alice", ?Y \rightarrow \_:\mathsf{a}, ?Z \rightarrow \mathsf{null}]$

Now, as opposed to [16], we define three notions of compatibility between substitutions:

- Two substitutions θ_1 and θ_2 are bravely compatible (*b-compatible*) when for all $x \in dom(\theta_1) \cap dom(\theta_2)$ either $x\theta_1 = \mathsf{null}$ or $x\theta_2 = \mathsf{null}$ or $x\theta_1 = x\theta_2$ holds. i.e., when $\theta_1 \cup \theta_2$ is a substitution over $dom(\theta_1) \cup dom(\theta_2)$.

- Two substitutions θ_1 and θ_2 are cautiously compatible (*c-compatible*) when they are b-compatible and for all $x \in dom(\theta_1) \cap dom(\theta_2)$ it holds that $x\theta_1 = x\theta_2$.

- Two substitutions θ_1 and θ_2 are strictly compatible (*s-compatible*) when they are c-compatible and for all $x \in dom(\theta_1) \cap dom(\theta_2)$ it holds that $x(\theta_1 \cup \theta_2) \neq \mathsf{null}$.

Analogously to [16] we define join, union, difference, and outer join between two sets of substitutions Ω_1 and Ω_2 over domains D_1 and D_2, respectively, all except union parameterized by $x \in \{\text{b,c,s}\}$:

$$\begin{aligned} \Omega_1 \bowtie_x \Omega_2 &= \{\theta_1 \cup \theta_2 \mid \theta_1 \in \Omega_1, \theta_2 \in \Omega_2, \text{ are } x\text{-compatible}\} \\ \Omega_1 \cup \Omega_2 &= \{\theta \mid \exists \theta_1 \in \Omega_1 \text{ with } \theta = \theta_1^{D_1 \cup D_2} \text{ or} \\ &\qquad\quad \exists \theta_2 \in \Omega_2 \text{ with } \theta = \theta_2^{D_1 \cup D_2}\} \\ \Omega_1 -_x \Omega_2 &= \{\theta \in \Omega_1 \mid \forall \theta_2 \in \Omega_2,\ \theta \text{ and } \theta_2 \text{ not } x\text{-compatible}\} \\ \Omega_1 ⟕_x \Omega_2 &= (\Omega_1 \bowtie_x \Omega_2) \cup (\Omega_1 -_x \Omega_2) \end{aligned}$$

The semantics of a graph pattern P over dataset $DS = (G, G_n)$, can now be defined recursively by the evaluation function returning sets of substitutions.

Definition 10 *(*Evaluation, extends [16, Def. 2]*) Let $t = (s, p, o)$ be a triple pattern, P, P_1, P_2 graph patterns, $DS = (G, G_n)$ a dataset, and $i \in G_n$, and $v \in Var$, then the x-joining evaluation $[[\cdot]]^x_{DS}$ is defined as follows:*

$$
\begin{array}{ll}
[[t]]^x_{DS} & = \{\theta \mid dom(\theta) = vars(P) \textit{ and } t\theta \in G\} \\
[[P_1 \textsf{ AND } P_2]]^x_{DS} & = [[P_1]]^x_{DS} \bowtie_x [[P_2]]^x_{DS} \\
[[P_1 \textsf{ UNION } P_2]]^x_{DS} & = [[P_1]]^x_{DS} \cup [[P_2]]^x_{DS} \\
[[P_1 \textsf{ MINUS } P_2]]^x_{DS} & = [[P_1]]^x_{DS} -_x [[P_2]]^x_{DS} \\
[[P_1 \textsf{ OPT } P_2]]^x_{DS} & = [[P_1]]^x_{DS} ⟕_x [[P_2]]^x_{DS} \\
[[\textsf{GRAPH } i\ P]]^x_{DS} & = [[P]]^x_{(i,\emptyset)} \\
[[\textsf{GRAPH } v\ P]]^x_{DS} & = \{\theta \cup [v \to g] \mid g \in G_n, \theta \in [[P[v \to g]\,]]^x_{(g,\emptyset)}\} \\
[[P \textsf{ FILTER } R]]^x_{DS} & = \{\theta \in [[P]]^x_{DS} \mid R\theta = \top\}
\end{array}
$$

Let R be a FILTER *expression, $u, v \in Var$, $c \in I \cup B \cup L$. The valuation of R on substitution θ, written $R\theta$ takes one of the three values $\{\top, \bot, \varepsilon\}$[6] and is defined as follows.*
$R\theta = \top$, if:

(1) $R =$ BOUND(v) with $v \in dom(\theta) \wedge v\theta \neq$ null;
(2) $R =$ isBLANK(v) with $v \in dom(\theta) \wedge v\theta \in B$;
(3) $R =$ isIRI(v) with $v \in dom(\theta) \wedge v\theta \in I$;
(4) $R =$ isLITERAL(v) with $v \in dom(\theta) \wedge v\theta \in L$;
(5) $R = (v{=}c)$ with $v \in dom(\theta) \wedge v\theta{=}c$;
(6) $R = (u{=}v)$ with $u, v \in dom(\theta) \wedge u\theta{=}v\theta \wedge u\theta{\neq}$null;
(7) $R = (\neg R_1)$ with $R_1\theta = \bot$;
(8) $R = (R1 \vee R2)$ with $R_1\theta = \top \vee R_2\theta = \top$;
(9) $R = (R1 \wedge R2)$ with $R_1\theta = \top \wedge R_2\theta = \top$.

$R\theta = \varepsilon$, if:

(1) $R =$ isBLANK(v), $R =$ isIRI(v), $R =$ isLITERAL(v),
or $R = (v{=}c)$ with $v \notin dom(\theta) \vee v\theta{=}$null;
(2) $R = (u{=}v)$ with $u \notin dom(\theta) \vee u\theta{=}$null
$\vee\, v \notin dom(\theta) \vee v\theta{=}$null;
(3) $R = (\neg R_1)$ and $R_1\theta = \varepsilon$;
(4) $R = (R_1 \vee R_2)$ and $(R_1\theta \neq \top \wedge R_2\theta \neq \top) \wedge$
$(R_1\theta = \varepsilon \vee R_2\theta = \varepsilon)$;
(5) $R = (R1 \wedge R2)$ and $R_1\theta = \varepsilon \vee R_2\theta = \varepsilon$.

$R\theta = \bot$ otherwise.

We will now exemplify the three different semantics defined above, namely bravely joining (b-joining), cautiously joining (c-joining), and strictly-joining (s-joining) semantics. When taking a closer look to the AND and MINUS operators, one will realize that all three semantics take a slightly differing view only when joining null. Indeed,

[6]$\top$ stands for "true", $\bot$ stands for "false" and ε stands for errors, see [18, Sec. 11.3] and Example 8 for details.

the AND operator behaves as the traditional natural join operator $\bowtie$ in relational algebra, when no null values are involved.

Take for instance, $DS = (\{\texttt{ex.org/bob}, \texttt{alice.org}\}, \emptyset)$ and $P = ((?X, \texttt{name}, ?Name)$ AND $(?X, \texttt{knows}, ?Friend))$. When viewing each solution set as a relational table with variables denoting attribute names, we can write:

?X	?Name
_:a	"Bob"
alice.org#me	"Alice"
_:c	"Bob"

$\bowtie$

?X	?Friend
_:a	_:b
alice.org#me	_:c

=

?X	?Name	?Friend
_:a	"Bob"	_:b
alice.org#me	"Alice"	_:c

Differences between the three semantics appear when joining over null-bound variables, as shown in the next example.

Example 9 *Let DS be as before and assume the following query which might be considered a naive attempt to ask for pairs of persons $?X1$, $?X2$ who share the same name and nickname where both, name and nickname are optional:*

$$P = (\ ((?X1, a, \texttt{Person})\ \textit{OPT}\ (?X1, \texttt{name}, ?N))\ \textit{AND}$$
$$((?X2, a, \texttt{Person})\ \textit{OPT}\ (?X2, \texttt{nick}, ?N))\ )$$

Again, we consider the tabular view of the resulting join:

?X1	?N
_:a	"Bob"
_:b	null
_:c	"Bob"
alice.org#me	"Alice"

$\bowtie_x$

?X2	?N
_:a	null
_:b	"Alice"
_:c	"Bobby"
alice.org#me	null

Now, let us see what happens when we evaluate the join $\bowtie_x$ with respect to the different semantics. The following result table lists in the last column which tuples belong to the result of b-, c- and s-join, respectively.

=

?X1	?N	X2	
_:a	"Bob"	_:a	b
_:a	"Bob"	alice.org#me	b
_:b	null	_:a	b,c
_:b	"Alice"	_:b	b
_:b	"Bobby"	_:c	b
_:b	null	alice.org#me	b,c
_:c	"Bob"	_:a	b
_:c	"Bob"	alice.org#me	b
alice.org#me	"Alice"	_:a	b
alice.org#me	"Alice"	_:b	b,c,s
alice.org#me	"Alice"	alice.org#me	b

Leaving aside the question whether the query formulation was intuitively broken, we remark that only the s-join would have the expected result. At the very least we might argue, that the liberal behavior of b-joins might be considered surprising in some cases. The c-joining semantics acts a bit more cautious in between the two, treating null *values as normal values, only unifiable with other* null *values.*

Compared to how joins over incomplete relations are treated in common relational database systems, the s-joining semantics might be considered the intuitive behavior. Another interesting divergence (which would rather suggest to adopt the c-joining semantics) shows up when we consider a simple idempotent join.

Example 10 *Let us consider the following single triple dataset* $DS = (\{(\texttt{alice.org\#me}, a, \texttt{Person})\}, \emptyset)$ *and the following simple query pattern:*

$P = ((?X, a, \texttt{Person})\ \textsf{\textit{UNION}}\ (?Y, a, \texttt{Person}))$

Clearly, this pattern, has the solution set

$[[P]]^x_{DS} = \{(\texttt{alice.org\#me}, \textsf{null}), (\textsf{null}, \texttt{alice.org\#me})\}$

under all three semantics. Surprisingly, $P' = (P\ \textsf{\textit{AND}}\ P)$ *has different solution sets for the different semantics. First,* $[[P']]^c_{DS} = [[P]]^x_{DS}$*, but* $[[P']]^s_{DS} = \emptyset$*, since* null *values are not compatible under the s-joining semantics. Finally,*

$[[P']]^b_{DS} = \{(\texttt{alice.org\#me}, \textsf{null}), (\textsf{null}, \texttt{alice.org\#me}),$
$(\texttt{alice.org\#me}, \texttt{alice.org\#me})\}$

As shown by this example, under the reasonable assumption, that the join operator is idempotent, i.e., $(P \bowtie P) \equiv P$, only the c-joining semantics behaves correctly.

However, the brave b-joining behavior is advocated by the current SPARQL document, and we might also think of examples where this obviously makes a lot of sense. Especially, when considering no explicit joins, but the implicit joins within the OPT operator:

Example 11 *Let* $DS = (\{\texttt{ex.org/bob}, \texttt{alice.org}\}, \emptyset)$ *and assume a slight variant of a query from [5] which asks for persons and some names for these persons, where preferably the* $\texttt{foaf}:\texttt{name}$ *is taken, and, if not specified,* $\texttt{foaf}:\texttt{nick}$*.*

$P = ((((?X, a, \texttt{Person})\ \textsf{\textit{OPT}}\ (?X, \texttt{name}, ?XNAME))$
$\textsf{\textit{OPT}}\ (?X, \texttt{nick}, ?XNAME))$

Only $[[P]]^b_{DS}$ *contains the expected solution* $(\_\texttt{:b}, "Alice")$ *for the bNode* $\_\texttt{:b}$*.*

All three semantics may be considered as variations of the original definitions in [16], for which the authors proved complexity results and various desirable features, such as semantics-preserving normal form transformations and compositionality. The following proposition shows that all these results carry over to the normative b-joining semantics:

Proposition 19 *Given a dataset* DS *and a pattern* P *which does not contain* ***GRAPH*** *patterns, the solutions of* $[[P]]_{DS}$ *as in [16] and* $[[P]]^b_{DS}$ *are in 1-to-1 correspondence.*

Proof. Given DS and P each substitution θ obtained by evaluation $[[P]]^b_{DS}$ can be reduced to a substitution θ' obtained from the evaluation $[[P]]_{DS}$ in [16] by dropping all mappings of the form $v \rightarrow \textsf{null}$ from θ. Likewise, each substitution θ' obtained from $[[P]]_{DS}$ can be extended to a substitution $\theta = \theta'^{vars(P)}$ for $[[P]]^b_{DS}$. □

Following the definitions from the SPARQL specification and [16], the b-joining semantics is the only admissible definition. There are still advantages for gradually defining alternatives towards traditional treatment of joins involving nulls. On the one hand, as we have seen in the examples above, the brave view on joining unbound variables might have partly surprising results, on the other hand, as we will see, the c- and s-joining semantics allow for a more efficient implementation in terms of Datalog rules.

Let us now take a closer look on some properties of the three defined semantics.

Compositionality and Equivalences As shown in [16], some implementations have a non-compositional semantics, leading to undesired effects such as non-commutativity of the join operator, etc. A semantics is called *compositional* if for each P' sub-pattern of P the result of evaluating P' can be used to evaluate P. Obviously, all three the c-, s- and b-joining semantics defined here retain this property, since all three semantics are defined recursively, and independent of the evaluation order of the sub-patterns.

The following proposition summarizes equivalences which hold for all three semantics, showing some interesting additions to the results of Pérez et al.

Proposition 20 (extends [16, Prop. 1]) *The following equivalences hold or do not hold in the different semantics as indicated after each law:*

(1) AND*,* UNION *are associative and commutative.* *(b,c,s)*
(2) $(P_1\ \mathsf{AND}\ (P_2\ \mathsf{UNION}\ P_3))$
$\equiv ((P_1\ \mathsf{AND}\ P_2)\ \mathsf{UNION}\ (P_1\ \mathsf{AND}\ P_3)).$ *(b)*
(3) $(P_1\ \mathsf{OPT}\ (P_2\ \mathsf{UNION}\ P_3))$
$\equiv ((P_1\ \mathsf{OPT}\ P_2)\ \mathsf{UNION}\ (P_1\ \mathsf{OPT}\ P_3)).$ *(b)*
(4) $((P_1\ \mathsf{UNION}\ P_2)\ \mathsf{OPT}\ P_3)$
$\equiv ((P_1\ \mathsf{OPT}\ P_3)\ \mathsf{UNION}\ (P_2\ \mathsf{OPT}\ P_3)).$ *(b)*
(5) $((P_1\ \mathsf{UNION}\ P_2)\ \mathsf{FILTER}\ R)$
$\equiv ((P_1\ \mathsf{FILTER}\ R)\ \mathsf{UNION}\ (P_2\ \mathsf{FILTER}\ R)).$ *(b,c,s)*
(6) AND *is idempotent, i.e.* $(P\ \mathsf{AND}\ P) \equiv P.$ *(c)*

Proof.[Sketch.] (1-5) for the b-joining semantics are proven in [16], (1): for c-joining and s-joining follows straight from the definitions. (2)-(4): the substitution sets $[[P_1]]^{c,s} = \{[?X \to a, ?Y \to b]\}$, $[[P_2]]^{c,s} = \{[?X \to a, ?Z \to c]\}$, $[[P_3]]^{c,s} = \{[?Y \to b, ?Z \to c]\}$ provide counterexamples for c-joining and s-joining semantics for all three equivalences (2)-(4). (5): The semantics of FILTER expressions and UNION is exactly the same for all three semantics, thus, the result for the b-joining semantics carries over to all three semantics. (6): follows from the observations in Example 10. □

Ideally, we would like to identify a subclass of programs, where the three semantics coincide. Obviously, this is the case for any query involving neither UNION nor OPT operators. Pérez et al. [16] define a bigger class of programs, including "well-behaving" optional patterns:

Definition 11 *(*[16, Def. 4]*) A* UNION*-free graph pattern* P *is* well-designed *if for every occurrence of a sub-pattern* $P' = (P_1\ \mathsf{OPT}\ P_2)$ *of* P *and for every variable* v *occurring in* P*, the following condition holds: if* v *occurs both in* P_2 *and outside* P' *then it also occurs in* P_1*.*

As may be easily verified by the reader, neither Example 9 nor Example 11, which are both UNION-free, satisfy the well-designedness condition. Since in the general case the equivalences for Prop. 20 do not hold, we also need to consider nested UNION patterns as a potential source for null bindings which might affect join results. We extend the notion of well-designedness, which direclty leads us to another correspondence in the subsequent proposition.

Definition 12 *A graph pattern P is* well-designed *if the condition from Def. 11 holds and for every occurrence of a sub-pattern $P' = (P_1 \textsf{ UNION } P_2)$ of P and for every variable v occurring in P', the following condition holds: if v occurs outside P' then it occurs in both P_1 and P_2.*

Proposition 21 *On well-designed graph patterns the c-, s-, and b-joining semantics coincide.*

Proof.[Sketch.] Follows directly from the observation that all variables which are re-used outside P' must be bound to a value unequal to null in P' due to well-designedness, and thus cannot generate null bindings which might carry over to joins. $\Box$

Likewise, we can identify "dangerous" variables in graph patterns, which might cause semantic differences:

Definition 13 *Let P' a sub-pattern of P of either the form $P' = (P_1 \textsf{OPT } P_2)$ or $P' = (P_1 \textsf{UNION } P_2)$. Any variable v in P' which violates the well-designedness-condition is called* possibly-null-binding *in P.*

Note that, so far we have only defined the semantics in terms of a pattern P and dataset DS, but not yet taken the result form V of query $Q = (V, P, DS)$ into account.

We now define *solution tuples* that were informally introduced in Sec. 2. Recall that by $\overline{V}$ we denote the tuple obtained from lexicographically ordering a set of variables in V. The notion $\overline{V}[V' \to \mathsf{null}]$ means that, after ordering V all variables from a subset $V' \subseteq V$ are replaced by null.

Definition 14 *(*Solution Tuples*) Let $Q = (V, P, DS)$ be a* SPARQL *query, and θ a substitution in $[[P]]^x_{DS}$, then we call the tuple $\overline{V}[(V \setminus vars(P)) \to \mathsf{null}]\theta$ a solution tuple of Q with respect to the x-joining semantics.*

Let us remark at this point, that as for the discussion of intuitivity of the different join semantics discussed in Examples 9–11, we did not yet consider combinations of different join semantics, e.g. using b-joins for OPT and c-joins for AND patterns. We leave this for further work.

3 Datalog and Answer Sets

In this paper we will use a very general form of Datalog commonly referred to as Answer Set Programming (ASP), i.e. function-free logic programming (LP) under the answer set semantics [1, 11]. ASP is widely proposed as a useful tool for various problem solving tasks in e.g. Knowledge Representation and Deductive databases. ASP extends Datalog with useful features such as negation as failure, disjunction in rule heads, aggregates [9], external predicates[8], etc. [7]

Let $Pred$, $Const$, Var, $exPr$ be sets of predicate, constant, variable symbols, and external predicate names, respectively. Note that we assume all these sets except $Pred$

[7]We consider ASP, more precisely a simplified version of ASP with so-called HEX-programs [8] here, since it is up to date the most general extension of Datalog.

and $Const$ (which may overlap), to be disjoint. In accordance with common notation in LP and the notation for external predicates from [7] we will in the following assume that $Const$ and $Pred$ comprise sets of numeric constants, string constants beginning with a lower case letter, or '`"`' quoted strings, and strings of the form ⟨quoted-string⟩^^⟨IRI⟩, ⟨quoted-string⟩@⟨valid-lang-tag⟩, Var is the set of string constants beginning with an upper case letter. Given $p \in Pred$ an *atom* is defined as $p(t_1, \ldots, t_n)$, where n is called the arity of p and $t_1, \ldots, t_n \in Const \cup Var$.

Moreover, we define a fixed set of external predicates $exPr = \{rdf, isBLANK, isIRI, isLITERAL, =, !=\}$ All external predicates have a fixed semantics and fixed arities, distinguishing *input* and *output terms*. The atoms $isBLANK[c](val)$, $isIRI[c](val)$, $isLITERAL[c](val)$ test the input term $c \in Const \cup Var$ (in square brackets) for being valid string representations of Blank nodes, IRI References or RDF literals, returning an output value $val \in \{\mathtt{t}, \mathtt{f}, \mathtt{e}\}$, representing truth, falsity or an error, following the semantics defined in [18, Sec. 11.3]. For the rdf predicate we write atoms as $rdf[i](s, p, o)$ to denote that $i \in Const \cup Var$ is an input term, whereas $s, p, o \in Const \cup Var$ are output terms which may be bound by the external predicate. The external atom $rdf[i](s, p, o)$ is true if (s, p, o) is an RDF triple *entailed* by the RDF graph which is accessibly at IRI i. For the moment, we consider simple RDF entailment [13] only. Finally, we write comparison atoms '$t_1 = t_2$' and '$t_1 \mathrel{!=} t_2$' in infix notation with $t_1, t_2 \in Const \cup Var$ and the obvious semantics of (lexicographic or numeric) (in)equality. Here, for $=$ either t_1 or t_2 is an output term, but at least one is an input term, and for $!=$ both t_1 and t_2 are input terms.

Definition 15 *Finally, a* rule *is of the form*

$$h \;\texttt{:-}\; b_1,\ \ldots,\ b_m,\ \texttt{not}\ b_{m+1},\ \ldots\ \texttt{not}\ b_n. \qquad (1)$$

where h and b_i ($1 \leq i \leq n$) are atoms, b_k ($1 \leq k \leq m$) are either atoms or external atoms, and `not` *is the symbol for negation as failure.*

We use $H(r)$ to denote the head atom h and $B(r)$ to denote the set of all body literals $B^+(r) \cup B^-(r)$ of r, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$.

The notion of input and output terms in external atoms described above denotes the binding pattern. More precisely, we assume the following condition which extends the standard notion of safety (cf. [21]) in Datalog with negation: Each variable appearing in a rule must appear in $B^+(r)$ in an atom or as an output term of an external atom.

Definition 16 *A* (logic) program Π *is defined as a set of safe rules r of the form (1).*

The *Herbrand base* of a program Π, denoted HB_Π, is the set of all possible ground versions of atoms and external atoms occurring in Π obtained by replacing variables with constants from $Const$, where we define for our purposes by $Const$ the union of the set of all constants appearing in Π as well as the literals, IRIs, and distinct constants for each blank node occurring in each RDF graph identified[8] by one of the IRIs in the (recursively defined) set I, where I is defined by the recursive closure of all

[8]By "identified" we mean here that IRIs denote network accessible resources which correspond to RDF graphs.

IRIs appearing in Π and all RDF graphs identified by IRIs in I.[9] As long as we assume that the Web is finite the grounding of a rule r, $ground(r)$, is defined by replacing each variable with the possible elements of HB_Π, and the grounding of program Π is $ground(\Pi) = \bigcup_{r \in \Pi} ground(r)$.

An *interpretation relative to* Π is any subset $\mathcal{I} \subseteq HB_\Pi$ containing only atoms. We say that $\mathcal{I}$ is a *model* of atom $a \in HB_\Pi$, denoted $\mathcal{I} \models a$, if $a \in \mathcal{I}$. With every external predicate name $\lg \in exPr$ with arity n we associate an $(n+1)$-ary Boolean function $f_{\lg}$ (called *oracle function*) assigning each tuple $(\mathcal{I}, t_1 \ldots, t_n)$ either 0 or 1. [10] We say that $\mathcal{I} \subseteq HB_\Pi$ is a *model* of a ground external atom $a = g[t_1, \ldots, t_m](t_{m+1}, \ldots, t_n)$, denoted $\mathcal{I} \models a$, iff $f_{\lg}(\mathcal{I}, t_1, \ldots, t_n) = 1$.

The semantics we use here generalizes the answer-set semantics [11][11], and is defined using the *FLP-reduct* [9], which is more elegant than the traditional GL-reduct [11] of stable model semantics and ensures minimality of answer sets also in presence of external atoms.

Let r be a ground rule. We define (i) $\mathcal{I} \models B(r)$ iff $\mathcal{I} \models a$ for all $a \in B^+(r)$ and $\mathcal{I} \not\models a$ for all $a \in B^-(r)$, and (ii) $\mathcal{I} \models r$ iff $I \models H(r)$ whenever $\mathcal{I} \models B(r)$. We say that $\mathcal{I}$ is a *model* of a program Π, denoted $\mathcal{I} \models \Pi$, iff $\mathcal{I} \models r$ for all $r \in ground(\Pi)$.

The *FLP-reduct* [9] of Π with respect to $\mathcal{I} \subseteq HB_\Pi$, denoted $\Pi^{\mathcal{I}}$, is the set of all $r \in ground(\Pi)$ such that $\mathcal{I} \models B(r)$. $\mathcal{I}$ is an *answer set of* Π iff $\mathcal{I}$ is a minimal model of $\Pi^{\mathcal{I}}$.

We did not consider further extensions common to many ASP dialects here, namely disjunctive rule heads, strong negation [11]. We note that for non-recursive programs, i.e. where the predicate dependency graph is acyclic, the answer set is unique. For the pure translation which we will give in Sec. 4 where we will produce such non-recursive programs from SPARQL queries, we could equally take other semantics such as the well-founded [10] semantics into account, which coincides with ASP on non-recursive programs.

4 From SPARQL to Datalog

We are now ready to define a translation from SPARQL to Datalog which can serve straightforwardly to implement SPARQL within existing rules engines. We start with a translation for c-joining semantics, which we will extend thereafter towards s-joining and b-joining semantics.

Translation Π^c_Q Let $Q = (V, P, DS)$, where $DS = (G, G_n)$ as defined above. We translate this query to a logic program Π^c_Q defined as follows.

[9]We assume the number of accessible IRIs finite.

[10]The notion of an oracle function reflects the intuition that external predicates compute (sets of) outputs for a particular input, depending on the interpretation. The dependence on the interpretation is necessary for instance for defining the semantics of external predicates querying OWL [8] or computing aggregate functions.

[11]In fact, we use slightly simplified definitions from [7] for HEX-programs, with the sole difference that we restrict ourselves to a fixed set of external predicates.

$$\begin{aligned}
\tau(V,(s,p,o),D,i) &= \texttt{answer}_{\texttt{i}}(\overline{V},D)\ \texttt{:-}\ \texttt{triple}(s,p,o,D). && (1)\\
\tau(V,(P'\ \mathsf{AND}\ P''),D,i) &= \tau(vars(P'),P',D,2*i)\ \cup\ \tau(vars(P''),P'',D,2*i+1)\ \cup\\
&\quad \texttt{answer}_{\texttt{i}}(\overline{V},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P')},D),\ \texttt{answer}_{\texttt{2*i+1}}((\overline{vars(P'')},D). && (2)\\
\tau(V,(P'\ \mathsf{UNION}\ P''),D,i) &= \tau(vars(P'),P',D,2*i)\ \cup\ \tau(vars(P''),P'',D,2*i+1)\ \cup\\
&\quad \texttt{answer}_{\texttt{i}}(\overline{V[(V\setminus vars(P'))\to \mathsf{null}]},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P')},D). && (3)\\
&\quad \texttt{answer}_{\texttt{i}}(\overline{V[(V\setminus vars(P''))\to \mathsf{null}]},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i+1}}(\overline{vars(P'')},D). && (4)\\
\tau(V,(P'\ \mathsf{MINUS}\ P''),D,i) &= \tau(vars(P'),P',D,2*i)\ \cup\ \tau(vars(P''),P'',D,2*i+1)\ \cup\\
&\quad \texttt{answer}_{\texttt{i}}(\overline{V[(V\setminus vars(P'))\to \mathsf{null}]},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P')},D), && (5)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{not}\ \texttt{answer}_{\texttt{2*i}}{}'(\overline{vars(P')\cap vars(P'')},D)\texttt{.}\\
&\quad \texttt{answer}_{\texttt{2*i}}{}'(\overline{vars(P')\cap vars(P'')},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i+1}}(\overline{vars(P'')},D).\ \} && (6)\\
\tau(V,(P'\ \mathsf{OPT}\ P''),D,i) &= \tau(V,(P'\ \mathsf{AND}\ P''),D,i)\ \cup\ \tau(V,(P'\ \mathsf{MINUS}\ P''),D,i)\\
\tau(V,(P\ \mathsf{FILTER}\ \ R),D,i) &= \tau(vars(P),P,D,2*i)\ \cup\\
&\quad LT(\texttt{answer}_{\texttt{i}}(\overline{V},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P)},D),R.) && (7)\\
\tau(V,(\mathsf{GRAPH}\ g\ \ P),D,i) &= \tau(V,P,g,i)\ \text{for}\ g\in V\cup I\\
&\quad \texttt{answer}_{\texttt{i}}(\overline{V},D)\ \texttt{:-}\ \texttt{answer}_{\texttt{i}}(\overline{V},g),\ \texttt{isIRI}(g),\ \texttt{not}\ \ g=\texttt{default}. && (8)
\end{aligned}$$

Alternate rules replacing (5)+(6):

$$\begin{aligned}
\texttt{answer}_{\texttt{i}}(\overline{V[(V\setminus vars(P'))\to \mathsf{null}]},D)\ &\texttt{:-}\ \ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P')},D),\ \texttt{not}\ \texttt{answer}_{\texttt{2*i}}{}'(\overline{vars(P')},D) && (5')\\
\texttt{answer}_{\texttt{2*i}}{}'(\overline{vars(P')},D)\ &\texttt{:-}\ \ \texttt{answer}_{\texttt{2*i}}(\overline{vars(P')},D),\ \texttt{answer}_{\texttt{2*i+1}}(\overline{vars(P'')},D). && (6')
\end{aligned}$$

Figure 2: Translation Π^c_Q from SPARQL queries semantics to Datalog.

$$\begin{aligned}
\Pi^c_Q =&\{\texttt{triple}(S,P,O,\texttt{default})\ \ \texttt{:-}\ \ \texttt{rdf}[d](S,P,O).\mid d\in G\}\\
&\cup\{\texttt{triple}(S,P,O,g)\ \ \texttt{:-}\ \ \texttt{rdf}[g](S,P,O).\ \mid\ g\in G_n\}\\
&\cup\,\tau(V,P,\texttt{default},1)
\end{aligned}$$

The first two rules serve to import the relevant RDF triples from the dataset into a 4-ary predicate `triple`. Under the dataset closedness assumption (see Def. 9) we may replace the second rule set, which imports the named graphs, by:

$$\texttt{triple}(S,P,O,G)\ \ \texttt{:-}\ \ \texttt{rdf}[G](S,P,O),HU(G),\texttt{isIRI}(G).$$

Here, the predicate HU stands for "Herbrand universe", where we use this name a bit sloppily, with the intention to cover all the relevant part of $\mathcal{C}$, recursively importing all possible IRIs in order to emulate the dataset closedness assumption. HU, can be computed recursively over the input triples, i.e.

$$\begin{aligned}
&HU(X)\ \texttt{:-}\ \texttt{triple}(X,P,O,D).\ \ HU(X)\ \texttt{:-}\ \texttt{triple}(S,X,O,D)\texttt{.}\\
&HU(X)\ \texttt{:-}\ \texttt{triple}(S,P,X,D).\ \ HU(X)\ \texttt{:-}\ \texttt{triple}(S,P,O,X).
\end{aligned}$$

The remaining program $\tau(V,P,\texttt{default},1)$ represents the actual query translation, where τ is defined recursively as shown in Fig. 2.

By $LT(\cdot)$ we mean the set of rules resulting from disassembling complex FILTER expressions (involving '$\neg$','$\wedge$','$\vee$') according to the rewriting defined by Lloyd and Topor [15] where we have to obey the semantics for errors, following Definition 10. In a nutshell, the rewriting $LT - rewrite(\cdot)$ proceeds as follows: Complex filters involving $\neg$ are transformed into negation normal form. Conjunctions of filter expressions are simply disassembled to conjunctions of body literals, disjunctions are handled by splitting the respective rule for both alternatives in the standard way. The resulting rules involve possibly negated atomic filter expressions in the bodies. Here, $BOUND(v)$ is translated to $v = \mathsf{null}$, $\neg BOUND(v)$ to $v! = \mathsf{null}$. $isBLANK(v)$, $isIRI(v)$, $isLITERAL(v)$ and their negated forms are replaced by their corresponding external atoms (see Sec. 3) $\mathsf{isBLANK}[v](\mathtt{t})$ or $\mathsf{isBLANK}[v](\mathtt{f})$, etc., respectively.

The resulting program Π^c_Q implements the c-joining semantics in the following sense:

Proposition 22 (Soundness and completeness of Π^c_Q) *For each atom of the form*

$$\mathtt{answer_1}(\vec{s}, \mathtt{default})$$

in the unique answer set M of Π^c_Q, $\vec{s}$ is a solution tuple of Q with respect to the c-joining semantics, and all solution tuples of Q are represented by the extension of predicate $\mathtt{answer_1}$ *in M.*

Without giving a proof, we remark that the result follows if we convince ourselves that $\tau(V, P, D, i)$ emulates exactly the recursive definition of $[[P]]^x_{DS}$. Moreover, together with Proposition 21, we obtain soundness and completeness of Π_Q for b-joining and s-joining semantics as well for well-designed query patterns.

Corollary 23 *For $Q = (V, P, DS)$, if P is well-designed, then the extension of predicate* $\mathtt{answer_1}$ *in the unique answer set M of Π^c_Q represents all and only the solution tuples for Q with respect to the x-joining semantics, for $x \in \{b, c, s\}$.*

Now, in order to obtain a proper translation for arbitrary patterns, we obviously need to focus our attention on the possibly-null-binding variables within the query pattern P. Let $vnull(P)$ denote the possibly-null-binding variables in a (sub)pattern P. We need to consider all rules in Fig. 2 which involve x-joins, i.e. the rules of the forms (2),(5) and (6). Since rules (5) and (6) do not make this join explicit, we will replace them by the equivalent rules (5') and (6') for Π^s_Q and Π^b_Q. The "extensions" to s-joining and b-joining semantics can be achieved by rewriting the rules (2) and (6'). The idea is to rename variables and add proper FILTER expressions to these rules in order to realize the b-joining and s-joining behavior for the variables in $V_N = vnull(P) \cap vars(P') \cap vars(P'')$.

Translation Π^s_Q The s-joining behavior can be achieved by adding FILTER expressions

$$R^s = (\bigwedge_{v \in V_N} BOUND(v))$$

to the rule bodies of (2) and (6'). The resulting rules are again subject to the LT-rewriting as discussed above for the rules of the form (7). This is sufficient to filter out any joins involving null values, thus achieving s-joining semantics, and we denote the program rewritten that way as Π^s_Q.

Translation Π^b_Q Obviously, b-joining semantics is more tricky to achieve, since we now have to relax the allowed joins in order to allow null bindings to join with *any* other value. We will again achieve this result by modifying rules (2) and (6') where we first do some variable renaming and then add respective FILTER expressions to these rules.

Step 1. We rename each variable $v \in V_N$ in the respective rule bodies to v' or v'', respectively, in order to disambiguate the occurrences originally from sub-pattern P' or P'', respectively. That is, for each rule (2) or (6'), we rewrite the body to:

$\texttt{answer}_{\texttt{2*i}}(\overline{vars(P')[V_N \rightarrow V'_N]}, D)$,
$\texttt{answer}_{\texttt{2*i+1}}(\overline{vars(P'')[V_N \rightarrow V''_N]}, D)$.

Step 2. We now add the following FILTER expressions $R^b_{(2)}$ and $R^b_{(6')}$, respectively, to the resulting rule bodies which "emulate" the relaxed b-compatibility:

$$\begin{aligned} R^b_{(2)} = \textstyle\bigwedge_{v \in VN}(\quad & ((v = v') \wedge (v' = v'')) \vee \\ & ((v = v') \ \wedge \ \neg BOUND(v'')) \vee \\ & ((v = v'') \ \wedge \ \neg BOUND(v')) \ ) \\ R^b_{(6')} = \textstyle\bigwedge_{v \in VN}(\quad & ((v = v') \wedge (v' = v'')) \vee \\ & ((v = v') \ \wedge \ \neg BOUND(v'')) \vee \\ & ((v = v') \ \wedge \ \neg BOUND(v')) \ ) \end{aligned}$$

The rewritten rules are again subject to the LT rewriting. Note that, strictly speaking the filter expression introduced here does not fulfill the assumption of safe filter expressions, since it creates new bindings for the variable v. However, these can safely be allowed here, since the translation only creates valid input/output term bindings for the external Datalog predicate '$=$'. The subtle difference between $R^b_{(2)}$ and $R^b_{(6')}$ lies in the fact that $R^b_{(2)}$ preferably "carries over" bound values from v' or v'' to v whereas $R^b_{(6')}$ always takes the value of v'. The effect of this becomes obvious in the translation of Example 11 which we leave as an exercise to the reader. We note that the potential exponential (with respect to $|V_N|$) blowup of the program size by unfolding the filter expressions into negation normal form during the LT rewriting[12] is not surprising, given the negative complexity results in [16].

In total, we obtain a program which Π^b_Q which reflects the normative b-joining semantics. Consequently, we get sound and complete query translations for all three semantics:

Corollary 24 (Soundness and completeness of Π^x_Q) *Given an arbitrary graph pattern P, the extension of predicate* $\texttt{answer}_1$ *in the unique answer set M of Π^x_Q represents all and only the solution tuples for $Q = (V, P, DS)$ with respect to the x-joining semantics, for $x \in \{b, c, s\}$.*

In the following, we will drop the superscript x in Π_Q implicitly refer to the normative b-joining translation/semantics.

[12]Lloyd and Topor can avoid this potential exponential blowup by introducing new auxiliary predicates. However, we cannot do the same trick, mainly for reasons of preserving safety of external predicates as defined in Sec. 3.

5 Possible Extensions

As it turns out, the embedding of SPARQL in the rules world opens a wide range of possibilities for combinations. In this section, we will first discuss some straightforward extensions of SPARQL which come practically for free with the translation to Datalog provided before. We will then discuss the use of SPARQL itself as a simple RDF rules language[13] which allows to combine RDF fact bases with implicitly specified further facts and discuss the semantics thereof briefly. We conclude this section with revisiting the open issue of entailment regimes covering RDFS or OWL semantics in SPARQL.

5.1 Additional Language Features

Set Difference As mentioned before, set difference is not present in the current SPARQL specification syntactically, though hidden, and would need to be emulated via a combination of OPTIONAL and FILTER constructs. As we defined the MINUS operator here in a completely modular fashion, it could be added straightforwardly without affecting the semantics definition.

Nested queries Nested queries are a distinct feature of SQL not present in SPARQL. We suggest a simple, but useful form of nested queries to be added: Boolean queries $Q_{\mathsf{ASK}} = (\emptyset, P_{\mathsf{ASK}}, DS_{\mathsf{ASK}}))$ with an empty result form (denoted by the keyword ASK) can be safely allowed within FILTER expressions as an easy extension fully compatible with our translation. Given query $Q = (V, P, DS)$, with sub-pattern $(P_1$ FILTER (ASK $Q_{\mathsf{ASK}}))$ we can modularly translate such subqueries by extending Π_Q with $\Pi_{Q'}$ where $Q' = (vars(P_1) \cap vars(P_{\mathsf{ASK}}), P_{\mathsf{ASK}}, DS_{\mathsf{ASK}}))$. Moreover, we have to rename predicate names $\mathtt{answer}_i$ to $\mathtt{answer}^{\mathtt{Q}'}{}_i$ in $\Pi_{Q'}$. Some additional considerations are necessary in order to combine this within arbitrary complex filter expressions, and we probably need to impose well-designedness for variables shared between P and P_{ASK} similar to Def. 12. We leave more details as future work.

5.2 Result Forms and Solution Modifiers

We have covered only SELECT queries so far. As shown in the previous section, we can consider ASK queries equally. A limited form of the CONSTRUCT result form, which allows to construct new triples could be emulated in our approach as well. Namely, we can allow queries of the form

$$Q_{\mathsf{C}} = (\mathsf{CONSTRUCT} P_{\mathsf{C}}, P, DS)$$

where P_{C} is a graph pattern consisting only of bNode-free triple patterns. We can model these by adding a rule

$$\mathtt{triple}(s, p, o, \mathsf{C}) \ \texttt{:-}\ \mathtt{answer}_1(\overline{vars(P_{\mathsf{C}})}, \mathtt{default}). \qquad (2)$$

[13]Thus, the "...(and back)" in the title of this paper!

to Π_Q for each triple (s, p, o) in P_{C}. The result graph is then naturally represented in the answer set of the program extended that way in the extension of the predicate `triple`.

5.3 SPARQL as a Rules Language

As it turns out with the extensions defined in the previous subsections, SPARQL itself may be viewed as an expressive rules language on top of RDF. CONSTRUCT statements have an obvious similarity with view definitions in SQL, and thus may be seen as rules themselves.

Intuitively, in the translation of CONSTRUCT we "stored" the new triples in a new triple outside the dataset DS. We can imagine a similar construction in order to define the semantics of queries over datasets mixing such CONSTRUCT statements with RDF data in the same turtle file.

Let us assume such a mixed file containing CONSTRUCT rules and RDF triples web-accessible at IRI g, and a query $Q = (V, P, DS)$, with $DS = (G, G_n)$. The semantics of a query over a dataset containing g may then be defined by recursively adding $\Pi_{Q_{\mathsf{C}}}$ to Π_Q for any CONSTRUCT query Q_{C} in g plus the rules (2) above with their head changed to $\texttt{triple}(s, p, o, g)$. We further need to add a rule

$$\texttt{triple}(s, p, o, default) \texttt{ :- } \texttt{triple}(s, p, o, g).$$

for each $g \in G$, in order not to omit any of the implicit triples defined by such "CONSTRUCT rules". Analogously to the considerations for nested ASK queries, we need to rename the $\texttt{answer}_i$ predicates and $default$ constants in every subprogram $\Pi_{Q_{\mathsf{C}}}$ defined this way.

Naturally, the resulting programs possibly involve recursion, and, even worse, recursion over negation as failure. Fortunately, the general answer set semantics, which we use, can cope with this. For some important aspects on the semantics of such distributed rules and facts bases, we refer to [17], where we also outline an alternative semantics based on the well-founded semantics. A more in-depth investigation of the complexity and other semantic features of such a combination is on our agenda.

5.4 Revisiting Entailment Regimes

The current SPARQL specification does not treat entailment regimes beyond RDF simple entailment. Strictly speaking, even RDF entailment is already problematic as a basis for SPARQL query evaluation; a simple query pattern like $P = (?X, \mathsf{rdf{:}type}, \mathsf{rdf{:}Property})$ would have infinitely many solutions even on the empty (sic!) dataset by matching the infinitely many axiomatic triples in the RDF(S) semantics.

Finite rule sets which approximate the RDF(S) semantics in terms of positive Datalog rules [17] have been implemented in systems like TRIPLE[14] or JENA[15]. Similarly, fragments and extensions of OWL [12, 3, 14] definable in terms of Datalog rule bases have been proposed in the literature. Such rule bases can be parametrically combined

[14]`http://triple.semanticweb.org/`
[15]`http://jena.sourceforge.net/`

with our translations, implementing what one might call RDFS$^-$ or OWL$^-$ entailment at least. It remains to be seen whether the SPARQL working group will define such reduced entailment regimes.

More complex issues arise when combining a nonmonotonic query language like SPARQL with ontologies in OWL. An embedding of SPARQL into a nonmonotonic rules language might provide valuable insights here, since it opens up a whole body of work done on combinations of such languages with ontologies [7, 19].

6 Conclusions & Outlook

In this paper, we presented three possible semantics for SPARQL based on [16] which differ mainly in their treatment of joins and their translations to Datalog rules. We discussed intuitive behavior of these different joins in several examples. As it turned out, the s-joining semantics which is close to traditional treatment of joins over incomplete relations and the c-joining semantics are nicely embeddable into Datalog. The b-joining semantics which reflects the normative behavior as described by the current SPARQL specification is most difficult to translate. We also suggested some extension of SPARQL, based on this translation. Further, we hope to have contributed to clarifying the relationships between the Query, Rules and Ontology layers of the Semantic Web architecture with the present work.

A prototype of the presented translation has been implemented on top of the dlvhex system, a flexible framework for developing extensions for the declarative Logic Programming Engine DLV[16]. The prototype is available as a plugin at `http://con.fusion.at/dlvhex/`. The web-page also provides an online interface for evaluation, where the reader can check translation results for various example queries, which we had to omit here for space reasons. We currently implemented the c-joining and b-joining semantics and we plan to gradually extend the prototype towards the features mentioned in Sec. 5, in order to query mixed RDF+SPARQL rule and fact bases. Implementation of further extensions, such as the integration of aggregates typical for database query language, and recently defined for recursive Datalog programs in a declarative way compatible with the answer set semantics [9], are on our agenda. We are currently not aware of any other engine implementing the full semantics defined in [16].

7 Acknowledgments

Special thanks go to Jos de Bruijn and Reto Krummenacher for discussions on earlier versions of this document, to Bijan Parsia, Jorge Pérez, and Andy Seaborne for valuable email-discussions, to Roman Schindlauer for his help on prototype implementation on top of dlvhex, and to the anonymous reviewers for various useful comments. This work is partially supported by the Spanish MEC under the project TIC-2003-9001 and by the EC funded projects TripCom (FP6-027324) and KnowledgeWeb (IST 507482).

[16]`http://www.dlvsystem.com/`

References

[1] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambr.Univ. Press, 2003.

[2] D. Beckett. Turtle - Terse RDF Triple Language. Tech. Report, 4 Apr. 2006.

[3] J. de Bruijn, A. Polleres, R. Lara, D. Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. In *Proc. WWW-2005*, 2005.

[4] J. Carroll, C. Bizer, P. Hayes, P. Stickler. Named graphs. *Journal of Web Semantics*, 3(4), 2005.

[5] R. Cyganiak. A relational algebra for sparql. Tech. Report HPL-2005-170, HP Labs, Sept. 2005.

[6] J. de Bruijn, E. Franconi, S. Tessaris. Logical reconstruction of normative RDF. *OWL: Experiences and Directions Workshop (OWLED-2005)*, 2005.

[7] T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, H. Tompits. Reasoning with rules and ontologies. *Reasoning Web 2006*, 2006. Springer

[8] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. *Int.l Joint Conf. on Art. Intelligence (IJCAI)*, 2005.

[9] W. Faber, N. Leone, G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. *Proc. of the 9th European Conf. on Art. Intelligence (JELIA 2004)*, 2004. Springer.

[10] A. V. Gelder, K. Ross, J. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *7th ACM Symp. on Principles of Database Systems*, 1988.

[11] M. Gelfond, V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[12] B. N. Grosof, I. Horrocks, R. Volz, S. Decker. Description logic programs: Combining logic programs with description logics. *Proc. WWW-2003*, 2003.

[13] P. Hayes. RDF semantics. *W3C Recommendation*, 10 Feb. 2004. `http://www.w3.org/TR/rdf-mt/`

[14] H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2), July 2005.

[15] J. W. Lloyd, R. W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.

[16] J. Pérez, M. Arenas, C. Gutierrez. Semantics and complexity of SPARQL. *The Semantic Web – ISWC 2006*, 2006. Springer.

[17] A. Polleres, C. Feier, A. Harth. Rules with contextually scoped negation. *Proc. 3rd European Semantic Web Conf. (ESWC2006)*, 2006. Springer.

[18] E. Prud'hommeaux, A. S. (ed.). SPARQL Query Language for RDF, *W3C Working Draft*, 4 Oct. 2006. `http://www.w3.org/TR/rdf-sparql-query/`

[19] R. Rosati. Reasoning with Rules and Ontologies. *Reasoning Web 2006*, 2006. Springer.

[20] SQL-99. Information Technology - Database Language SQL- Part 3: Call Level Interface (SQL/CLI). Technical Report INCITS/ISO/IEC 9075-3, INCITS/ISO/IEC, Oct. 1999. Standard specification.

[21] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

Published in Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007), pp. 878–896, Nov. 2007, Springer LNCS vol. 3803

SPARQL++ for Mapping between RDF Vocabularies*

Axel Polleres[†] *François Scharffe*[‡] *Roman Schindlauer*[§ ¶]

[†] *DERI Galway, National University of Ireland, Galway*
[‡] *Leopold-Franzens Universität Innsbruck, Austria*
[§] *Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy*
[¶] *Institut für Informationssysteme, Technische Universität Wien Favoritenstraße 9-11, A-1040 Vienna, Austria*

Abstract

Lightweight ontologies in the form of RDF vocabularies such as SIOC, FOAF, vCard, etc. are increasingly being used and exported by "serious" applications recently. Such vocabularies, together with query languages like SPARQL also allow to syndicate resulting RDF data from arbitrary Web sources and open the path to finally bringing the Semantic Web to operation mode. Considering, however, that many of the promoted lightweight ontologies overlap, the lack of suitable standards to describe these overlaps in a declarative fashion becomes evident. In this paper we argue that one does not necessarily need to delve into the huge body of research on ontology mapping for a solution, but SPARQL itself might — with extensions such as external functions and aggregates — serve as a basis for declaratively describing ontology mappings. We provide the semantic foundations and a path towards implementation for such a mapping language by means of a translation to Datalog with external predicates.

1 Introduction

As RDF vocabularies like SIOC,[1] FOAF,[2] vCard,[3] etc. are increasingly being used and exported by "serious" applications we are getting closer to bringing the Semantic

*This research has been partially supported by the European Commission under the FP6 projects inContext (IST-034718), REWERSE (IST 506779), and Knowledge Web (FP6-507482), by the Austrian Science Fund (FWF) under project P17212-N04, as well as by Science Foundation Ireland under the Lion project (SFI/02/CE1/I131).

[1]`http://sioc-project.org/`
[2]`http://xmlns.com/foaf/0.1/`
[3]`http://www.w3.org/TR/vcard-rdf`

Web to operation mode. The standardization of languages like RDF, RDF Schema and OWL has set the path for such vocabularies to emerge, and the recent advent of an operable query language, SPARQL, gave a final kick for wider adoption. These ingredients allow not only to publish, but also to syndicate and reuse metadata from arbitrary distibuted Web resources in flexible, novel ways.

When we take a closer look at emerging vocabularies we realize that many of them overlap, but despite the long record of research on ontology mapping and alignment, a standard language for defining mapping rules between RDF vocabularies is still missing. As it turns out, the RDF query language SPARQL [27] itself is a promising candidate for filling this gap: Its CONSTRUCT queries may themselves be viewed as rules over RDF. The use of SPARQL as a rules language has several advantages: (i) the community is already familiar with SPARQL's syntax as a query language, (ii) SPARQL supports already a basic set of built-in predicates to filter results and (iii) SPARQL gives a very powerful tool, including even non-monotonic constructs such as OPTIONAL queries.

When proposing the use of SPARQL's CONSTRUCT statement as a rules language to define mappings, we should first have a look on existing proposals for syntaxes for rules languages on top of RDF(S) and OWL. For instance, we can observe that SPARQL may be viewed as syntactic extension of SWRL [19]: A SWRL rule is of the form $ant \Rightarrow cons$, where both antecedent and consequent are conjunctions of atoms $a_1 \land \ldots \land a_n$. When reading these conjunctions as basic graph patterns in SPARQL we might thus equally express such a rule by a CONSTRUCT statement:

$$\text{CONSTRUCT } \{ \; cons \; \} \text{ WHERE } \{ \; ant \; \}$$

In a sense, such SPARQL "rules" are more general than SWRL, since they may be evaluated on top of arbitrary RDF data and — unlike SRWL — not only on top of valid OWL DL. Other rules language proposals, like WRL [8] or TRIPLE [9] which are based on F-Logic [22] Programming may likewise be viewed to be layerable on top of RDF, by applying recent results of De Bruijn et al. [6, 7]. By the fact that (i) expressive features such as negation as failure which are present in some of these languages are also available in SPARQL[4] and (ii) F-Logic molecules in rule heads may be serialized in RDF again, we conjecture that rules in these languages can similarly be expressed as syntactic variants of SPARQL CONSTRUCT statements.[5]

On the downside, it is well-known that even a simple rules language such as SWRL already lead to termination/undecidability problems when mixed with ontology vocabulary in OWL without care. Moreover, it is not possible to express even very simple mappings between common vocabularies such as FOAF [5] and VCard [20] in SPARQL only. In order to remedy this situation, we propose the following approach to enable complex mappings over ontologies: First, we keep the expressivity of the underlying ontology language low, restricting ourselves to RDFS, or, more strictly speaking to, ρdf^- [24] ontologies; second, we extend SPARQL's CONSTRUCT by features which are almost essential to express various mappings, namely: a set of useful built-in functions (such as string-concatenation and arithmetic functions on numeric literal values) and aggregate functions (min, max, avg). Third, we show that evaluating

[4]see [27, Section 11.4.1]

[5]with the exception of predicates with arbitrary arities

SPARQL queries on top of ρdf$^-$ ontologies plus mapping rules is decidable by translating the problem to query answering over HEX-programs, i.e., logic programs with external built-ins using the answer-set semantics, which gives rise to implementations on top of existing rules engines such as dlvhex. A prototype of a SPARQL engine for evaluating queries over combined datasets consisting of ρdf$^-$ and SPARQL mappings has been implemented and is avaiblable for testing online.[6]

The remainder of this paper is structured as follows. We start with some motivating examples of mappings which can and can't be expressed with SPARQL CONSTRUCT queries in Section 2 and suggest syntactic extensions of SPARQL, which we call SPARQL++, in order to deal with the mappings that go beyond. In Section 3 we introduce HEX-programs, whereafter in Section 4 we show how SPARQL++ CONSTRUCT queries can be translated to HEX-programs, and thereby bridge the gap to implementations of SPARQL++. Next, we show how additional ontological inferences by ρdf$^-$ ontologies can be itself viewed as a set of SPARQL++ CONSTRUCT "mappings" to HEX-programs and thus embedded in our overall framework, evaluating mappings and ontological inferences at the same level, while retaining decidability. After a brief discussion of our current prototype and a discussion of related approaches, we conclude in Section 6 with an outlook to future work.

2 Motivating Examples – Introducing SPARQL

Most of the proposals in the literature for defining mappings between ontologies use subsumption axioms (by relating defining classes or (sub)properties) or bridge rules [3]. Such approaches do not go much beyond the expressivity of the underlying ontology language (mostly RDFS or OWL). Nonetheless, it turns out that these languages are insufficient for expressing mappings between even simple ontologies or when trying to map actual sets of data from one RDF vocabulary to another one.

In Subsection 10.2.1 of the latest SPARQL specification [27] an example for such a mapping from FOAF [5] to VCard [20] is explicitly given, translating the VCard properties into the respective FOAF properties most of which could equally be expressed by simple rdfs:subPropertyOf statements. However, if we think the example a bit further, we quickly reach the limits of what is expressible by subclass- or subproperty statements.

Example 12 *A simple and straightforward example for a mapping from* `VCard:FN` *to* `foaf:name` *is given by the following* SPARQL *query:*

```
CONSTRUCT { ?X foaf:name ?FN . } WHERE { ?X VCard:FN ?FN . FILTER isLiteral(?FN) }
```

The filter expression here reduces the mapping by a kind of additional "type checking" where only those names are mapped which are not fully specified by a substructure, but merely given as a single literal.

Example 13 *The situation quickly becomes more tricky for other terms, as for instance mapping between* `VCard:n` *(name) and* `foaf:name`*, because* `VCard:n` *consists of*

[6] `http://kr.tuwien.ac.at/research/dlvhex/`

a substructure consisting of Family name*,* Given name*,* Other names*,* honorific Prefixes*, and* honorific Suffixes*. One possibility is to concatenate all these to constitute a* `foaf:name` *of the respective person or entity:*

```
CONSTRUCT { ?X foaf:name ?Name . }
WHERE { ?X VCard:N ?N .
        OPTIONAL {?N VCard:Family ?Fam } OPTIONAL {?N VCard:Given  ?Giv }
        OPTIONAL {?N VCard:Other  ?Oth } OPTIONAL {?N VCard:Prefix ?Prefix }
        OPTIONAL {?N VCard:Suffix ?Suffix }
        FILTER (?Name = fn:concat(?Prefix," ",?Giv, " ",?Fam," ",?Oth," ",?Suffix))
      }
```

We observe the following problem here: First, we use filters for constructing a new binding which is not covered by the current SPARQL *specification, since filter expressions are not meant to create new bindings of variables (in this case the variable* `?Name`*), but only filter existing bindings. Second, if we wanted to model the case where e.g., several other names were provided, we would need built-in functions beyond what the current* SPARQL *spec provides, in this case a string manipulation function such as* `fn:concat`*.* SPARQL *provides a subset of the functions and operators defined by XPath/XQuery, but these cover only boolean functions, like arithmetic comparison operators or regular expression tests and basic arithmetic functions. String manipulation routines are beyond the current spec. Even if we had the full range of XPath/XQuery functions available, we would admittedly have to also slightly "extend"* `fn:concat` *here, assuming that unbound variables are handled properly, being replaced by an empty string in case one of the optional parts of the name structure is not defined.*

Apart from built-in functions like string operations, aggregate functions such as count, minimum, maximum or sum, are another helpful construct for many mappings that is currently not available in SPARQL.

Finally, although we can query and create new RDF graphs by SPARQL CONSTRUCT statements mapping one vocabulary to another, there is no well-defined way to combine such mappings with arbitrary data, especially when we assume that (1) mappings are not restricted to be unidirectional from one vocabulary to another, but bidirectional, and (2) additional ontological inferences such as subclass/subproperty relations defined in the mutually mapped vocabularies should be taken into account when querying over syndicated RDF data and mappings. We propose the following extensions of SPARQL:

- We introduce an extensible set of useful built-in and aggregate functions.
- We permit function calls and aggregates in the CONSTRUCT clause,
- We further allow CONSTRUCT queries nested in FROM statements, or more general, allowing CONSTRUCT queries as part of the dataset.

2.1 Built-in Functions and Aggregates in Result Forms

Considering Example 12, it would be more intuitive to carry out the string translation from `VCard:n` to `foaf:name` in the result form, i.e., in the CONSTRUCT clause:

```
CONSTRUCT {?X foaf:name fn:concat(?Prefix," ",?Giv," ",?Fam," ",?Oth," ",?Suffix).}
WHERE { ?X VCard:N ?N .
        OPTIONAL {?N VCard:Family ?Fam } OPTIONAL {?N VCard:Given  ?Giv }
        OPTIONAL {?N VCard:Other  ?Oth } OPTIONAL {?N VCard:Prefix ?Prefix }
        OPTIONAL {?N VCard:Suffix ?Suffix } }
```

Another example for a non-trivial mapping is the different treatment of telephone numbers in FOAF and VCard.

Example 14 *A* `VCard:tel` *is a* `foaf:phone` *– more precisely,* `VCard:tel` *is related to* `foaf:phone` *as follows. We have to create a URI from the RDF literal value defining* `vCard:tel` *here, since vCard stores Telephone numbers as string literals, whereas FOAF uses resources, i.e., URIs with the* `tel:` *URI-scheme:*

```
CONSTRUCT { ?X foaf:phone rdf:Resource(fn:concat("tel:",fn:encode-for-uri(?T)) . }
WHERE { ?X VCard:tel ?T . }
```

Here we assumed the availability of a cast-function, which converts an `xs:string` to an RDF resource. While the distinction between literals and URI references in RDF usually makes perfect sense, this example shows that conversions between URI references and literals become necessary by practical uses of RDF vocabularies.

The next example shall illustrate the need for aggregate functions in mappings.

Example 15 *The DOAP vocabulary [10] contains revision, i.e. version numbers of released versions of projects. With an aggregate function* MAX*, one can map DOAP information into the RDF Open Source Software Vocabulary [32], which talks about the latest release of a project, by picking the maximum value (numerically or lexicographically) of the set of revision numbers specified by a graph pattern as follows:*

```
CONSTRUCT { ?P os:latestRelease MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project . }
```

Here, the WHERE *clause singles out all projects, while the aggregate selects the highest (i.e., latest) revision date of any available version for that project.*

2.2 Nested CONSTRUCT Queries in FROM Clauses

The last example show another example of "aggregation" which is not possible with SPARQL upfront, but may be realized by nesting CONSTRUCT queries in the FROM clause of a SPARQL query.

Example 16 *Imagine you want to map/infer from an ontology having co-author relationships declared using* `dc:creator` *properties from the Dublin Core metadata vocabulary to* `foaf:knows`*, i.e., you want to specify*

If $?a$ and $?b$ have co-authored the same paper, then $?a$ knows $?b$.

The problem here is that a mapping using CONSTRUCT *clauses needs to introduce new blank nodes for both $?a$ and $?b$ (since* `dc:creator` *is a datatype property usually just giving the name string of the author) and then need to infer the knows relation, so what we really want to express is a mapping*

If $?a$ and $?b$ are `dc:creators` of the same paper, then someone named with `foaf:name` $?a$ `foaf:knows` someone with `foaf:name` $?b$.

A first-shot solution could be:

```
CONSTRUCT { _:a foaf:knows _:b . _:a foaf:name ?n1 . _:b foaf:name ?n2 . }
FROM <g>  WHERE { ?p dc:creator ?n1 . ?p dc:creator ?n2 . FILTER ( ?n1 != ?n2 ) }
```

Let us consider the present paper as example graph g:

```
g: <http://ex.org/papers#sparqlmappingpaper> dc:creator "Axel"
   <http://ex.org/papers#sparqlmappingpaper> dc:creator "Roman"
   <http://ex.org/papers#sparqlmappingpaper> dc:creator "Francois"
```

By the semantics of blank nodes in CONSTRUCT clauses — SPARQL creates new blank node identifiers for each solutions set matching the WHERE clause — the above would infer the following additional triples:

```
 _:a1 foaf:knows _:b1.  _:a1 foaf:name  "Axel".     _:b1 foaf:name  "Roman".
 _:a2 foaf:knows _:b2.  _:a2 foaf:name  "Axel".     _:b2 foaf:name  "Francois".
 _:a3 foaf:knows _:b3.  _:a3 foaf:name  "Francois". _:b3 foaf:name  "Roman".
 _:a4 foaf:knows _:b4.  _:a4 foaf:name  "Francois". _:b4 foaf:name  "Axel".
 _:a5 foaf:knows _:b5.  _:a5 foaf:name  "Roman".    _:b5 foaf:name  "Axel".
 _:a6 foaf:knows _:b6.  _:a6 foaf:name  "Roman".    _:b6 foaf:name  "Francois".
```

Obviously, we lost some information in this mapping, namely the corellations that the "Axel" knowing "Francois" is the same "Axel" that knows "Roman", etc. We could remedy this situation by allowing to nest CONSTRUCT queries in the FROM clause of SPARQL *queries as follows:*

```
  CONSTRUCT { ?a knows ?b . ?a foaf:name ?aname . ?b foaf:name ?bname . }
  FROM { CONSTRUCT { _:auth foaf:name ?n . ?p aux:hasAuthor _:auth . }
         FROM <g> WHERE { ?p dc:creator ?n . } }
  WHERE { ?p aux:hasAuthor ?a . ?a foaf:name ?aname .
          ?p aux:hasAuthor ?b . ?b foaf:name ?bname . FILTER ( ?a != ?b ) }
```

Here, the "inner" CONSTRUCT creates a graph with unique blank nodes for each author per paper, whereas the outer CONSTRUCT then aggregates a more appropriate answer graph, say:

```
_:auth1 foaf:name "Axel". _:auth2 foaf:name "Roman". _:auth3 foaf:name "Francois".
_:auth1 foaf:knows _:auth2. _:auth1 foaf:knows _:auth3.
_:auth2 foaf:knows _:auth1. _:auth2 foaf:knows _:auth3.
_:auth3 foaf:knows _:auth1. _:auth3 foaf:knows _:auth2.
```

In the following, we will extend SPARQL syntactically and semantically to deal with these features. This extended version of the language, which we call SPARQL++ shall allow to evaluate SPARQL queries on top of RDF(S) data combined with mappings again expressed in SPARQL++.

3 Preliminaries – HEX-Programs

To evaluate SPARQL++ queries, we will translate them to so-called HEX-*programs* [12], an extension of logic programs under the answer-set semantics.

Let $Pred$, $Const$, Var, $exPr$ be mutually disjoint sets of predicate, constant, variable symbols, and external predicate names, respectively. In accordance with common notation in LP and the notation for external predicates from [11] we will in the following assume that $Const$ comprises the set of numeric constants, string constants beginning with a lower case letter, or double-quoted string literals, and IRIs.[7] Var is the set of string constants beginning with an uppercase letter. Elements from $Const \cup Var$ are called *terms*. Given $p \in Pred$ an *atom* is defined as $p(t_1, \ldots, t_n)$, where n is called the arity of p and $t_1, \ldots, t_n$ are terms. An *external atom* is of the form

$$g[Y_1, \ldots, Y_n](X_1, \ldots, X_m),$$

where $Y_1, \ldots, Y_n$ is a list of predicates and terms and $X_1, \ldots, X_m$ is a list of terms (called *input list* and *output list*, respectively), and $g \in exPr$ is an external predicate name. We assume the *input* and *output arities* n and m fixed for g. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates and terms. Note that this means that external predicates, unlike usual definitions of built-ins in logic programming, can not only take constant parameters but also (extensions of) predicates as input.

Definition 17 *A* rule *is of the form*

$$h \leftarrow b_1, \ldots, b_m, not\, b_{m+1}, \ldots not\, b_n \tag{1}$$

where h and b_i ($m+1 \leq i \leq n$) are atoms, b_k ($1 \leq k \leq m$) are either atoms or external atoms, and 'not' is the symbol for negation as failure.

We use $H(r)$ to denote the head atom h and $B(r)$ to denote the set of all body literals $B^+(r) \cup B^-(r)$ of r, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$.

The notion of input and output terms in external atoms described above denotes the binding pattern. More precisely, we assume the following condition which extends the standard notion of safety (cf. [31]) in Datalog with negation.

Definition 18 (Safety) *Each variable appearing in a rule must appear in a non-negated body atom or as an output term of an external atom.*

Finally, we define HEX-programs.

Definition 19 *A* HEX-program *P is defined as a set of safe rules r of the form* (*1*)*.*

The *Herbrand base* of a HEX-program P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $Const$. The grounding of a rule r, $ground(r)$, is defined accordingly, and the grounding of program P is $ground(P) = \bigcup_{r \in P} ground(r)$.

An *interpretation relative to P* is any subset $I \subseteq HB_P$ containing only atoms. We say that I is a *model* of atom $a \in HB_P$, denoted $I \models a$, if $a \in I$. With every external predicate name $e \in exPr$ we associate an $(n+m+1)$-ary Boolean function f_e (called *oracle function*) assigning each tuple $(I, y_1 \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where n/m are the input/output arities of e, $I \subseteq HB_P$, $x_i \in Const$, and $y_j \in Pred \cup Const$.

[7]For the purpose of this paper, we will disregard language-tagged and datatyped literals in the translation to HEX-programs.

We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = e[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, iff $f_e(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in \mathit{ground}(P)$.

The semantics we use here generalizes the answer-set semantics [16] and is defined using the *FLP-reduct* [15], which is more elegant than the traditional Gelfond-Lifschitz reduct of stable model semantics and ensures minimality of answer sets also in presence of external atoms: The *FLP-reduct* of P with respect to $I \subseteq HB_P$, denoted P^I, is the set of all $r \in \mathit{ground}(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of P^I.

By the *cautious extension* of a predicate p we denote the set of atoms with predicate symbol p in the intersection of all answer sets of P.

For our purposes, we define a fixed set of external predicates $\mathit{exPr} = \{\mathit{rdf}, \mathit{isBLANK}, \mathit{isIRI}, \mathit{isLITERAL}, =, \,!\!=, \mathit{REGEX}, \mathit{CONCAT}, \mathit{COUNT}, \mathit{MAX}, \mathit{MIN}, \mathit{SK}\}$ with a fixed semantics as follows. We take these external predicates as examples, which demonstrate the HEX-programs are expressive enough to model all the necessary ingredients for evaluating SPARQL filters ($\mathit{isBLANK}, \mathit{isIRI}, \mathit{isLITERAL}, =, \,!\!=, \mathit{REGEX}$) and also for more expressive built-in functions and aggregates ($\mathit{CONCAT}, \mathit{SK}, \mathit{COUNT}, \mathit{MAX}, \mathit{MIN}$). Here, we take CONCAT just as an example built-in, assuming that more XPath/XQuery functions could similarly be added.

For the rdf predicate we write atoms as $\mathit{rdf}[i](s, p, o)$ to denote that $i \in \mathit{Const} \cup \mathit{Var}$ is an input term, whereas $s, p, o \in \mathit{Const}$ are output terms which may be bound by the external predicate. The external atom $\mathit{rdf}[i](s, p, o)$ is true if (s, p, o) is an RDF triple *entailed* by the RDF graph which is accessibly at IRI i. For the moment, here we consider simple RDF entailment [18] only.

The atoms $\mathit{isBLANK}[c](\mathit{val})$, $\mathit{isIRI}[c](\mathit{val})$, $\mathit{isLITERAL}[c](\mathit{val})$ test input term $c \in \mathit{Const} \cup \mathit{Var}$ (in square brackets) for being a valid string representation of a blank node, IRI reference or RDF literal. The atom $\mathit{REGEX}[c_1, c_2](\mathit{val})$ test whether c_1 matches the regular expression given in c_2. All these external predicates return an output value $\mathit{val} \in \{\mathtt{t}, \mathtt{f}, \mathtt{e}\}$, representing truth, falsity or an error, following the semantics defined in [27, Sec. 11.3].

We write comparison atoms '$t_1 = t_2$' and '$t_1 \,!\!= t_2$' in shorthand infix notation with $t_1, t_2 \in \mathit{Const} \cup \mathit{Var}$ and the obvious semantics of (lexicographic or numeric) (in)equality. Here, for $=$ either t_1 or t_2 is an output term, but at least one is an input term, and for $!\!=$ both t_1 and t_2 are input terms.

Apart from these truth-valued external atoms we add other external predicates which mimic built-in functions an aggregates. As an example predicate for a built-in, we chose the predicate $\mathit{CONCAT}[c_1, \ldots, c_n](c_{n+1})$ with variable input arity which concatenates string constants $c_1, \ldots, c_n$ into c_{n+1} and thus implements the semantics of `fn:concat` in XPath/XQuery [23].

Next, we define external predicates which mimic aggregate functions over a certain predicate. Let $p \in \mathit{Pred}$ with arity n, and $x_1, \ldots, x_n \in \mathit{Const} \cup \{\mathit{mask}\}$ where mask is a special constant not allowed to appear anywhere except in input lists of aggregate predicates.

Then $COUNT[p, x_1, \ldots, x_n](c)$ is true if c equals the number of distinct tuples $(t_1, \ldots, t_n)$, such that $I \models p(t_1, \ldots, t_n)$ and for all x_i different from the constant $mask$ it holds that $t_i = x_i$.

$MAX[p, x_1, \ldots, x_n](c)$ (and $MIN[p, x_1, \ldots, x_n](c)$, resp.) is true if among all tuples $(t_1, \ldots, t_n)$, such that $I \models p(t_1, \ldots, t_n)$, c is the lexicographically greatest (smallest, resp.) value among all the t_i such that $x_i = mask$.[8]

We will illustrate the use of these external predicates to express aggregations in Section 4.4 below when discussing the actual translation from SPARQL++ to HEX-programs.

Finally, the external predicate $SK[id, v_1, \ldots, v_n](sk_{n+1})$ computes a unique, new "Skolem"-like term $id(v_1, \ldots, v_n)$ from its input parameters. We will use this built-in function in our translation of SPARQL queries with blank nodes in the CONSTRUCT part. Similar to the aggregate functions mentioned before, when using SK we will need to take special care in our translation in order to retain strong safety.

As widely known, for programs without external predicates, safety guarantees that the number of entailed ground atoms is finite. Though, by external atoms in rule bodies, new, possibly infinitly many, ground atoms could be generated, even if all atoms themselves are safe. In order to avoid this, a stronger notion of safety for HEX-programs is defined in [30]: Informally, this notion says that a HEX-program is *strongly safe*, if no external predicate recursively depends on itself, thus defining a notion of stratification over external predicates. Strong safety guarantees finiteness of models as well as finite computability of external atoms.

4 Extending SPARQL towards mappings

In Section 2 we have shown that an extension of the CONSTRUCT clause is needed for SPARQL to be suitable for mapping tasks. In the following, we will formally define extended SPARQL queries which allow to integrate built-in functions and aggregates in CONSTRUCT clauses as well as in FILTER expressions. We will define the semantics of such extended queries, and, moreover, we will provide a translation to HEX-programs, building upon an existing translation presented in [26].

A SPARQL++ *query* $Q = (R, P, DS)$ consists of a result form R, a graph pattern P, and an extended dataset DS as defined below.[9] We refer to [27] for syntactical details and will explain these in the following as far as necessary.

For a SELECT query, a result form R is simply a set of variables, whereas for a CONSTRUCT query, the result form R is a set of triple patterns.

We assume the pairwise disjoint, infinite sets I, B, L and Var, which denote IRIs, blank node identifiers, RDF literals, and variables respectively. $I \cup L \cup Var$ is also called the set of *basic RDF terms*. In this paper, we allow as blank node identifiers nested ground terms similar to HiLog terms [4], such that B is defined recursively over an infinite set of constant blank node identifiers B_c as follows:

[8]Note that in this definition we allow min/max to aggregate over several variables.

[9]As we deal mainly with CONSTRUCT queries here, we will ignore solution modifiers.

- each element of B_c is a blank node identifier, i.e., $B_c \subseteq B$.
- for $b \in B$ and $t_1, \ldots, t_n$ in $I \cup B \cup L$, $b(t_1, \ldots, t_n) \in B$.

Now, we extend the SPARQL syntax by allowing built-in functions and aggregates in place of basic RDF terms in graph patterns (and thus also in CONSTRUCT clauses) as well as in filter expressions. We define the set *Blt* of *built-in terms* as follows:

- All basic terms are built-in terms.
- If *blt* is a built-in predicate (e.g., `fn:concat` from above or another XPath/XQuery functions), and $c_1, \ldots, c_n$ are built-in terms then $blt(c_1, \ldots, c_n)$ is a built-in term.
- If agg is an aggregate function (e.g., $COUNT$, MIN, MAX), P a graph pattern, and V a tuple of variables appearing in P, then $agg\ (V : P)$ is a built-in term.[10]

In the following we will introduce extended graph patterns that may include built-in terms and extended datasets that can be constituted by CONSTRUCT queries.

4.1 Extended Graph Patterns

As for *graph patterns*, we follow the recursive definition from [25]:

- a triple pattern (s, p, o) is a graph pattern where $s, o \in Blt$ and $p \in I \cup Var$.[11] Triple patterns which only contain basic terms are called *basic triple patterns* and *value-generating triple patterns* otherwise.
- if P, P_1 and P_2 are graph patterns, $i \in I \cup Var$, and R a filter expression then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, $(P_1 \text{ UNION } P_2)$, $(\text{GRAPH } i\ P)$, and $(P \text{ FILTER } R)$ are graph patterns.[12]

For any pattern P, we denote by $vars(P)$ the set of all variables occurring in P and by $\overline{vars}(P)$ the tuple obtained by the lexicographic ordering of all variables in P. As *atomic filter expression*, we allow here the unary predicates BOUND (possibly with variables as arguments), isBLANK, isIRI, isLITERAL, and binary equality predicates '$=$' with arbitrary safe built-in terms as arguments. *Complex filter expressions* can be built using the connectives '$\neg$', '$\wedge$', and '$\vee$'.

Similar to aggregates in logic programming, we use a notion of safety. First, given a query $Q = (R, P, DS)$ we allow only basic triple patterns in P, ie. we only allow built-ins and aggregates only in FILTERs or in the result pattern R. Second, a built-in term *blt* occurring in the result form or in P in a query $Q = (R, P, DS)$ is *safe* if all variables recursively appearing in *blt* also appear in a basic triple pattern within P .

[10] This aggregate syntax is adapted from the resp. definition for aggregates in LP from [15].
[11] We do not consider blanks nodes here as these can be equivalently replaced by variables [6].
[12] We use AND to keep with the operator style of [25] although it is not explicit in SPARQL.

4.2 Extended Datasets

In order to allow the definition of RDF data side-by-side with implicit data defined by mappings of different vocabularies or, more general, views within RDF, we define an *extended RDF graph* as a set of RDF triples $I \cup L \cup B \times I \times I \cup L \cup B$ and CONSTRUCT queries. An RDF graph (or dataset, resp.) without CONSTRUCT queries is called a *basic graph* (or dataset, resp.).

The dataset $DS = (G, \{(g_1, G_1), \ldots (g_k, G_k)\})$ of a SPARQL query is defined by (i) a default graph G, i.e., the RDF merge [18, Section 0.3] of a set of extended RDF graphs, plus (ii) a set of named graphs, i.e., pairs of IRIs and corresponding extended graphs.

Without loss of generality (there are other ways to define the dataset such as in a SPARQL protocol query), we assume DS defined by the IRIs given in a set of FROM and FROM NAMED clauses. As an exception, we assume that any CONSTRUCT query which is part of an extended graph G by default (i.e., in the absence of FROM and FROM NAMED clauses) has the dataset $DS = (G, \emptyset)$ For convenience, we allow extended graphs consisting of a single CONSTRUCT statement to be written directly in the FROM clause of a SPARQL++ query, like in Example 16.

We will now define syntactic restrictions on the CONSTRUCT queries allowed in extended datasets, which retain finite termination on queries over such datasets. Let G be an extended graph. First, for any CONSTRUCT query $Q = (R, P, DS_Q)$ in G, DS_Q we allow only triple patterns $tr = (s, p, o)$ in P or R where $p \in I$, i.e., neither blank nodes nor variables are allowed in predicate positions in extended graphs, and, additionally, $o \in I$ for all triples such that $p = \texttt{rdf:type}$. Second, we define a predicate-class-dependency graph over an extended dataset $DS = (G, \{(g_1, G_1), \ldots (g_k, G_k)\})$ as follows. The predicate-class-dependency graph for DS has an edge $p \to r$ with $p, r \in I$ for any CONSTRUCT query $Q = (R, P, DS)$ in G with r (or p, resp.) either (i) a predicate different from $\texttt{rdf:type}$ in a triple in R (or P, resp.), or (ii) an object in an $\texttt{rdf:type}$ triple in R (or P, resp.). All edges such that r occurs in a value-generating triple are marked with '$*$'. We now say that DS is *strongly safe* if its predicate-class-dependency graph does not contain any cycles involving marked edges. As it turns out, in our translation in Section 4.4 below, this condition is sufficient (but not necessary) to guarantee that any query can be translated to a strongly safe HEX-program.

Like in [29] we assume that blank node identifiers in each query $Q = (R, P, DS)$ have been standardized apart, i.e., that no blank nodes with the same identifiers appear in a different scope. The scope of a blank node identifier is defined as the graph or graph pattern it appears in, where each WHERE or CONSTRUCT clause open a "fresh" scope . For instance, take the extended graph dataset in Fig. 1(a), its standardized apart version is shown in Fig. 1(b). Obviously, extended datasets can always be standardized apart in linear time in a preprocessing step.

4.3 Semantics

The semantics of SPARQL++ is based on the formal semantics for SPARQL queries by Pérez et al. in [25] and its translation into HEX-programs in [26].

```
g1:  :paper2 foaf:maker _:a.
     _:a foaf:name "Jean Deau".

g2: :paper1 dc:creator "John Doe".
    :paper1 dc:creator "Joan Dough".
    CONSTRUCT {_:a foaf:knows _:b .
               _:a foaf:name ?N1 .
               _:b foaf:name ?N2 . }
    WHERE {?X dc:creator ?N1,?N2.
           FILTER( ?N1 != ?N2 ) }
```

(a)

```
g1:  :paper2 foaf:maker _:b1.
     _:b1 foaf:name "Jean Deau".

g2: :paper1 dc:creator "John Doe".
    :paper1 dc:creator "Joan Dough".
    CONSTRUCT {_:b2 foaf:knows _:b3 .
               _:b2 foaf:name ?N1 .
               _:b3 foaf:name ?N2 . }
    WHERE {?X dc:creator ?N1,?N2.
           FILTER( ?N1 != ?N2 ) }
```

(b)

Figure 1: Standardizing apart blank node identifiers in extended datasets.

We denote by T_{null} the union $I \cup B \cup L \cup \{\mathsf{null}\}$, where null is a dedicated constant denoting the unknown value not appearing in any of I, B, or L, how it is commonly introduced when defining outer joins in relational database systems. A *substitution* θ from Var to T_{null} is a partial function $\theta : Var \rightarrow T_{\mathsf{null}}$. We write substitutions in postfix notation in square brackets, i.e., if $t, t' \in Blt$ and $v \in Var$, then $t[v/t']$ is the term obtained from replacing all occcurences of v in t by t'. The *domain* of θ, denoted by $dom(\theta)$, is the subset of Var where θ is defined. The lexicographic ordering of this subset is denoted by $\overline{dom}(Var)$. For a substitution θ and a set of variables $D \subseteq Var$ we define the substitution θ^D with domain D as follows

$$x\theta^D = \begin{cases} x\theta & \text{if } x \in dom(\theta) \cap D \\ \mathsf{null} & \text{if } x \in D \setminus dom(\theta) \end{cases}$$

Let $x \in Var$, θ_1, θ_2 be substitutions, then $\theta_1 \cup \theta_2$ is the substitution obtained as follows:

$$x(\theta_1 \cup \theta_2) = \begin{cases} & x\theta_1 & \text{if } x\theta_1 \text{ defined and } x\theta_2 \text{ undefined} \\ \text{else:} & x\theta_1 & \text{if } x\theta_1 \text{ defined and } x\theta_2 = \mathsf{null} \\ \text{else:} & x\theta_2 & \text{if } x\theta_2 \text{ defined} \\ \text{else:} & \text{undefined} & \end{cases}$$

Thus, in the union of two substitutions defined values in one take precedence over null values the other substitution. Two substitutions θ_1 and θ_2 are *compatible* when for all $x \in dom(\theta_1) \cap dom(\theta_2)$ either $x\theta_1 = \mathsf{null}$ or $x\theta_2 = \mathsf{null}$ or $x\theta_1 = x\theta_2$ holds, i.e., when $\theta_1 \cup \theta_2$ is a substitution over $dom(\theta_1) \cup dom(\theta_2)$. Analogously to Pérez et al. we define join, union, difference, and outer join between two sets of substitutions Ω_1 and Ω_2 over domains D_1 and D_2, respectively:

$$\begin{aligned} \Omega_1 \bowtie \Omega_2 &= \{\theta_1 \cup \theta_2 \mid \theta_1 \in \Omega_1, \theta_2 \in \Omega_2, \theta_1 \text{ and } \theta_2 \text{ are compatible}\} \\ \Omega_1 \cup \Omega_2 &= \{\theta \mid \exists \theta_1 \in \Omega_1 \text{ with } \theta = \theta_1^{D_1 \cup D_2} \text{ or} \\ &\qquad \exists \theta_2 \in \Omega_2 \text{ with } \theta = \theta_2^{D_1 \cup D_2}\} \\ \Omega_1 - \Omega_2 &= \{\theta \in \Omega_1 \mid \text{ for all } \theta_2 \in \Omega_2, \ \theta \text{ and } \theta_2 \text{ not compatible}\} \\ \Omega_1 \sqsupset\!\!\bowtie \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 - \Omega_2) \end{aligned}$$

Next, we define the application of substitutions to built-in terms and triples: For a built-in term t, by $t\theta$ we denote the value obtained by applying the substitution to all variables in t. By $eval_\theta(t)$ we denote the value obtained by (i) recursively evaluating all built-in and aggregate functions, and (ii) replacing all bNode identifiers by complex bNode identifiers according to θ, as follows:

$eval_\theta(\texttt{fn:concat}(c_1, c_2, \ldots, c_n))$	Returns the `xs:string` that is the concatenation of the values of $c_1\theta$,...,$c_1\theta$ after conversion. If any of the arguments is the empty sequence or null, the argument is treated as the zero-length string.
$eval_\theta(\texttt{COUNT}(V : P))$	Returns the number of distinct[13]answer substitutions for the query $Q = (V, P\theta, DS)$ where DS is the dataset of the encapsulating query.
$eval_\theta(\texttt{MAX}(V : P))$	Returns the maximum (numerically or lexicographically) of distinct answer substitutions for the query $Q = (V, P\theta, DS)$.
$eval_\theta(\texttt{MIN}(V : P))$	Analogous to `MAX`, but returns the minimum.
$eval_\theta(t)$	Returns $t\theta$ for all $t \in I \cup L \cup Var$, and $t(\overline{dom}(\theta)\theta)$ for $t \in B$.[14]

Finally, for a triple pattern $tr = (s, p, o)$ we denote by $tr\theta$ the triple $(s\theta, p\theta, o\theta)$, and by $eval_\theta(tr)$ the triple $(eval_\theta(s), eval_\theta(p), eval_\theta(o))$.

The evaluation of a graph pattern P over a basic dataset $DS = (G, G_n)$, can now be defined recursively by sets of substitutions, extending the definitions in [25, 26].

Definition 20 *Let $tr = (s, p, o)$ be a basic triple pattern, P, P_1, P_2 graph patterns, and $DS = (G, G_n)$ a basic dataset, then the* evaluation $[\![\cdot]\!]_{DS}$ *is defined as follows:*

$$
\begin{array}{ll}
[\![tr]\!]_{DS} & = \{\theta \mid dom(\theta) = vars(P) \textit{ and } tr\theta \in G\} \\
[\![P_1 \textsf{ AND } P_2]\!]_{DS} & = [\![P_1]\!]_{DS} \bowtie [\![P_2]\!]_{DS} \\
[\![P_1 \textsf{ UNION } P_2]\!]_{DS} & = [\![P_1]\!]_{DS} \cup [\![P_2]\!]_{DS} \\
[\![P_1 \textsf{ OPT } P_2]\!]_{DS} & = [\![P_1]\!]_{DS} \sqsupset\!\!\bowtie [\![P_2]\!]_{DS} \\
[\![\textsf{GRAPH } i\ P]\!]_{DS} & = [\![P]\!]_{(i,\emptyset)}, \textit{for } i \in G_n \\
[\![\textsf{GRAPH } v\ P]\!]_{DS} & = \{\theta \cup [v/g] \mid g \in G_n \textit{ and } \theta \in [\![P[v/g]\,]\!]_{(g,\emptyset)}\}, \textit{for } v \in Var \\
[\![P \textsf{ FILTER } R]\!]_{DS} & = \{\theta \in [\![P]\!]_{DS} \mid R\theta = \top\}
\end{array}
$$

Let R be a filter expression, $u, v \in Blt$. The valuation of R on a substitution θ, written $R\theta$ takes one of the three values $\{\top, \bot, \varepsilon\}$[15] and is defined as follows.

$$
\begin{array}{lll}
R\theta = \top, \textit{ if:} & (1) & R = \textsf{BOUND}(v) \textit{ with } v \in dom(\theta) \ \wedge\ eval_\theta(v) \neq \textsf{null}; \\
& (2) & R = \textsf{isBLANK}(v) \textit{ with } eval_\theta(v) \in B; \\
& (3) & R = \textsf{isIRI}(v) \textit{ with } eval_\theta(v) \in I; \\
& (4) & R = \textsf{isLITERAL}(v) \textit{ with } eval_\theta(v) \in L; \\
& (5) & R = (u = v) \textit{ with } eval_\theta(u) = eval_\theta(v) \wedge\ eval_\theta(u) \neq \textsf{null}; \\
& (6) & R = (\neg R_1) \textit{ with } R_1\theta = \bot; \\
& (7) & R = (R1 \vee R2) \textit{ with } R_1\theta = \top\ \vee\ R_2\theta = \top; \\
& (8) & R = (R1 \wedge R2) \textit{ with } R_1\theta = \top\ \wedge\ R_2\theta = \top. \\
R\theta = \varepsilon, \textit{ if:} & (1) & R = \textsf{isBLANK}(v), R = \textsf{isIRI}(v), R = \textsf{isLITERAL}(v), \textit{ or} \\
& & R = (u = v) \textit{ with } (v \in Var \wedge v \notin dom(\theta))\ \vee\ eval_\theta(v) = \textsf{null} \vee \\
& & \qquad\qquad (u \in Var \wedge u \notin dom(\theta))\ \vee\ eval_\theta(u) = \textsf{null}; \\
& (2) & R = (\neg R_1) \textit{ and } R_1\theta = \varepsilon; \\
& (3) & R = (R_1 \vee R_2) \textit{ and } R_1\theta \neq \top\ \wedge\ R_2\theta \neq \top\ \wedge\ (R_1\theta = \varepsilon\ \vee\ R_2\theta = \varepsilon); \\
& (4) & R = (R1 \wedge R2) \textit{ and } R_1\theta = \varepsilon\ \vee\ R_2\theta = \varepsilon. \\
R\theta = \bot \textit{ otherwise.} & &
\end{array}
$$

In [26] we have shown that the semantics defined this way corresponds with the original semantics for SPARQL defined in [25] without complex built-in and aggregate terms and on basic datasets.[16]

Note that, so far we have only defined the semantics in terms of a pattern P and basic dataset DS, but neither taken the result form R nor extended datasets into account.

[13]Note that we give a set based semantics to the counting built-in, we do not take into account duplicate solutions which can arise from the multi-set semantics in [27] when counting.

[14]For blank nodes $eval_\theta$ constructs a new blank node identifier, similar to Skolemization.

[16]Our definition here only differs in in the application of $eval_\theta$ on built-in terms in filter expressions which does not make a difference if only basic terms appear in FILTERs.

As for the former, we proceed with formally define solutions for SELECT and CONSTRUCT queries, respectively. The semantics of a SELECT query $Q = (V, P, DS)$ is fully determined by its *solution tuples* [26].

Definition 21 *Let $Q = (R, P, DS)$ be a* SPARQL++ *query, and θ a substitution in $[\![P]\!]_{DS}$, then we call the tuple $\overline{vars(P)}\theta$ a solution tuple of Q.*

As for a CONSTRUCT queries, we define the *solution graphs* as follows.

Definition 22 *Let $Q = (R, P, DS)$ be a* SPARQL CONSTRUCT *query where blank node identifiers in DS and R have been standardized apart and $R = \{t_1, \ldots, t_n\}$ is the result graph pattern. Further, for any $\theta \in [\![P]\!]_{DS}$, let $\theta' = \theta^{vars(R) \cup vars(P)}$. The solution graph for Q is then defined as the triples obtained from*

$$\bigcup_{\theta in [\![P]\!]_{DS}} \{eval_{\theta'}(t_1), \ldots, eval_{\theta'}(t_n)\}$$

by eliminating all non-valid RDF triples.[17]

Our definitions so far only cover basic datasets. Extended datasets, which are implicitly defined bring the following additional challenges: (i) it is not clear upfront which blank node identifiers to give to blank nodes resulting from evaluating CONSTRUCT clauses, and (ii) extended datasets might involve recursive CONSTRUCT definitions which construct new triples in terms of the same graph in which they are defined. As for (i), we remedy the situation by constructing new identifier names via a kind of Skolemization, as defined in the function $eval_\theta$, see the table on page 82. $eval_\theta$ generates a unique blank node identifier for each solution θ. Regarding (ii) we avoid possibly infinite datasets over recursive CONSTRUCT clauses by the strong safety restriction in Section 4.2. Thus, we can define a translation from extended datasets to HEX-programs which uniquely identifies the solutions for queries over extended datasets.

4.4 Translation to HEX-Programs

Our translation from SPARQL++ queries to HEX-programs is based on the translation for non-extended SPARQL queries outlined in [26]. Similar to the well-known correspondence between SQL and Datalog, SPARQL++ queries can be expressed by HEX-programs, which provide the additional machinery necessary for importing and processing RDF data as well as evaluating built-ins and aggregates. The translation consists of two basic parts: (i) rules that represent the query's graph pattern (ii) rules defining the triples in the extended datasets.

We have shown in [26] that solution tuples for any query Q can be generated by a logic program and are represented by the extension of a designated predicate $\texttt{answer}_Q$, assuming that the triples of the dataset are available in a predicate $\texttt{triple}_Q$. We refer to [26] for details and only outline the translation here by examples.

Complex graph patterns can be translated recursively in a rather straightforward way, where unions and join of graph patterns can directly be expressed by appropriate rule constructions, whereas OPTIONAL patterns involve negation as failure.

[17]That is, triples with null values or blank nodes in predicate position, etc.

Example 17 *Let query q select all persons who do not know anybody called "John Doe" from the extended dataset $DS = (g1 \cup g2, \emptyset)$, i.e., the merge of the graphs in Fig. 1(b).*

```
SELECT  ?P FROM <g1> FROM <g2>
WHERE { ?P rdf:type foaf:Agent . FILTER ( !BOUND(?P1) )
        OPTIONAL { P? foaf:knows ?P1 . ?P1 foaf:name "John Doe" . } }
```

This query can be translated to the following HEX*-program:*

```
answer_q(P) :- answer1_q(P,P1), P1 = null.
answer1_q(P,P1) :- answer2_q(P), answer_q3(P,P1).
answer1_q(P,null) :- answer2_q(P), not answer3_q'(P).
answer2_q(P) :- triple_q(P,rdf:type,foaf:Agent,def).
answer3_q(P,P1) :- triple_q(P,foaf:knows,P1,def),triple(P1,foaf:name,"John Doe",def).
answer3_q'(P) :- answer3_q(P,P1).
```

More complex queries with nested patterns can be translated likewise by introducing more auxiliary predicates. The program part defining the $\mathtt{triple}_q$ predicate fixes the triples of the dataset, by importing all explicit triples in the dataset as well as recursively translating all CONSTRUCT clauses and subqueries in the extended dataset.

Example 18 *The program to generate the dataset triples for the extended dataset $DS = (g1 \cup g2, \emptyset)$ looks as follows:*

```
triple_q(S,P,O,def) :- rdf["g1"](S,P,O).
triple_q(S,P,O,def) :- rdf["g2"](S,P,O).
triple_q(B2,foaf:knows,B3,def) :- SK[b2(X,N1,N2)](B2),SK[b3(X,N1,N2)](B3),
                                  answer_C1,g2(X,N1,N2).
triple_q(B2,foaf:name,N1,def) :- SK[b2(X,N1,N2)](B2), answer_C1,g2(X,N1,N2).
triple_q(B3,foaf:knows,N2,def) :- SK[b3(X,N1,N2)](B3), answer_C1,g2(X,N1,N2).
answer_C1,g2(X,N1,N2) :- triple_q(X,dc:creator, N1,def),
                         tripleq(X,dc:creator,N2,def), N1 != N2.
```

The first two rules import all triples given explicitly in graphs $g1, g2$ by means of the "standard" RDF import HEX *predicate. The next three rules create the triples from the CONSTRUCT in graph $g2$, where the query pattern is translated by an own subprogram defining the predicate $\mathtt{answer}_{C_1,g2}$, which in this case only consists of a single rule.*

The example shows the use of the external function SK to create blank node ids for each solution tuple as mentioned before, which we need to emulate the semantics of blank nodes in CONSTRUCT statements.

Next, we turn to the use of HEX aggregate predicates in order to translate aggregate terms. Let $Q = (R, P, DS)$ and $a = agg\,(V : P_a)$ – here, $V \subseteq vars(P_a)$ is the tuple of variables we want to aggregate over – be an aggregate term appearing either in R or in a filter expression in P. Then, the idea is that a can be translated by an external atom $agg[aux, \overline{vars}(P_a)'[V/mask]\,](v_a)$ where

(i) $\overline{vars}(P_a)'$ is obtained from $\overline{vars}(P_a)$ by removing all the variables which only appear in P_a but not elsewhere in P,
(ii) the variable v_a takes the place of a,
(iii) aux_a is a new predicate defined by a rule: $aux_a(\overline{vars}(P_a)') \leftarrow answer_a(\overline{vars}(P_a))$.
(iv) $answer_a$ is the predicate defining the solution set of the query $Q_a = (vars(P_a), P_a, DS)$

Example 19 *The following rules mimic the CONSTRUCT query of Example 15:*

```
triple(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                  triple(P,rdf:type,doap:Project,gr).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def), triple(R,doap:revision,V,def).
```

With the extensions the translation in [26] outlined here for extended datasets, aggregate and built-in terms we can define the solution tuples of an SPARQL++ query $Q = (R, P, DS)$ over an extended dataset now as precisely the set of tuples corresponding to the cautious extension of the predicate $\texttt{answer}_q$.

4.5 Adding ontological inferences by encoding ρdf$^-$ into SPARQL

Trying the translation sketched above on the query in Example 17 we observe that we would not obtain any answers, as no triples in the dataset would match the triple pattern `?P rdf:type foaf:Agent` in the WHERE clause. This still holds if we include the vocabulary definition of FOAF at `http://xmlns.com/foaf/spec/index.rdf` to the dataset, since the machinery introduced so far could not draw any additional inferences from the triple `foaf:maker rdfs:range foaf:Agent` which would be necessary in order to figure out that Jean Deau is indeed an agent. There are several open issues on using SPARQL on higher entailment regimes than simple RDF entailment which allow such inferences. One such problem is the presences of infinite axiomatic triples in RDF semantics or several open compatibility issues with OWL semantics, see also [11]. However, we would like to at least add some of the inferences of the RDFS semantics. To this end, we will encode an small but very useful subset of RDFS, called ρdf [24] into the extended dataset. ρdf, defined by Muñoz et al., restricts the RDF vocabulary to its essentials by only focusing on the properties rdfs:subPropertyOf, rdfs:subClassOf,rdf:type, rdfs:domain, and rdfs:range, ignoring other constituents of the RDFS vocabulary. Most importantly, Muñoz et al. prove that (i) ρdf entailment corresponds to full RDF entailment on graphs not mentioning RDFS vocabulary outside ρdf, and (ii) that ρdf entailment can be reduced to five axiomatic triples (concerned with reflexivity of the subproperty relationship) and 14 entailment rules. Note that for graphs which do not mention subclass or subproperty relationships, which is usually the case for patterns in SPARQL queries or the mapping rules we encode here, even a reflexive-relaxed version of ρdf that does not contain any axiomatic triples is sufficient. We can write down all but one of the entailment rules of reflexive-relaxed ρdf as CONSTRUCT queries which we consider implicitly present in the extended dataset:

```
CONSTRUCT {?A :subPropertyOf ?C} WHERE {?A :subPropertyOf ?B. ?B :subPropertyOf ?C.}
CONSTRUCT {?A :subClassOf ?C} WHERE { ?A :subClassOf ?B. ?B :subClassOf ?C. }
CONSTRUCT {?X ?B ?Y}       WHERE { ?A :subPropertyOf ?B. ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :subClassOf ?B. ?X rdf:type ?A. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :domain ?B. ?X ?A ?Y. }
CONSTRUCT {?Y rdf:type ?B} WHERE { ?A :range ?B.  ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :domain ?B. ?C :subPropertyOf ?A. ?X ?C ?Y.}
CONSTRUCT {?Y rdf:type ?B} WHERE { ?A :range ?B.  ?C :subPropertyOf ?A. ?X ?C ?Y.}
```

There is one more entailment rule for reflexive-relaxed ρdf concerning that blank node renaming preserves ρdf entailment. However, it is neither straightforwardly possible, nor desirable to encode this by CONSTRUCTs like the other rules. Blank node renaming might have unintuitive effects on aggregations and in connection with OPTIONAL queries. In fact, keeping blank node identifiers in recursive CONSTRUCTs after standardizing apart is what keeps our semantics finite, so we skip this rule, and call the resulting ρdf fragment encoded by the above CONSTRUCTs ρdf^-. Some care is in order concerning strong safety of the resulting dataset when adding ρdf^-. To still ensure strong safety of the translation, we complete the predicate-class-dependency graph by additional edges between all pairs of resources connected by subclassOf or subPropertyOf, domain, or range relations and checking the same safety condition as before on the graph extended in this manner.

4.6 Implementation

We implemented a prototype of a SPARQL++ engine based on on the HEX-program solver dlvhex.[18] The prototype exploits the rewriting mechanism of the dlvhex framework, taking care of the translation of a SPARQL++ query into the appropriate HEX-program, as laid out in Section 4.4. The system implements external atoms used in the translation, namely (i) the RDF atom for data import, (ii) the aggregate atoms, and (iii) a string concatenation atom implementing both the $CONCAT$ function and the SK atom for bNode handling. The engine can directly be fed with a SPARQL++ query. The default syntax of a dlvhex results corresponds to the usual answer format of logic programming engines, i.e., sets of facts, from which we generate an XML representation, which can subsequently be transformed easily to a valid RDF syntax by an XSLT to export solution graphs.

5 Related work

The idea of using SPARQL CONSTRUCT queries is in fact not new, even some implemented systems such as TopBraid Composer already seem to offer this feature, [19] however without a defined and layered semantics, and lacking aggregates or built-ins, thus insufficient to express mappings such as the ones studied in this article.

Our notion of extended graphs and datasets generalizes so-called networked graphs defined by Schenk and Staab [29] who also use SPARQL CONSTRUCT statements as rules with a slightly different motivation: dynamically generating views over graphs. The authors only permit bNode- and built-in free CONSTRUCTs whereas we additionally allow bNodes, built-ins and aggregates, as long as strong safety holds which only restricts recursion over value-generating triples. Another differenece is that their semantics bases on the well-founded instead of the stable model semantics.

PSPARQL [1], a recent extension of SPARQL, allows to query RDF graphs using regular path expressions over predicates. This extension is certainly useful to represent mappings and queries over graphs. We conjecture that we can partly emulate such path expressions by recursive CONSTRUCTs in extended datasets.

As an interesting orthogonal approach, we mention iSPARQL [21] which proposes an alternative way to add external function calls to SPARQL by introducing so called

[18]Available with dlvhex on `http://www.kr.tuwien.ac.at/research/dlvhex/`.

[19]`http://composing-the-semantic-web.blogspot.com/2006/09/ontology-mapping-with-sparql-construct.html`

virtual triple patterns which query a "virtual" dataset that could be an arbitrary service. This approach does not need syntactic extensions of the language. However, an implementation of this extension makes it necessary to know upfront which predicates denote virtual triples. The authors use their framework to call a library of similarity measure functions but do not focus on mappings or CONSTRUCT queries.

As already mentioned in the introduction, other approaches often allow only mappings at the level of the ontology level or deploy their own rules language such as SWRL [19] or WRL [8]. A language more specific for ontology mapping is C-OWL [3], which extends OWL with bridge rules to relate ontological entities. C-OWL is a formalism close to distributed description logics [2]. These approaches partially cover aspects which we cannot handle, e.g., equating instances using owl:sameAs in SWRL or relating ontologies based on a local model semantics [17] in C-OWL. None of these approaches though offers aggregations which are often useful in practical applications of RDF data syndication, the main application we target in the present work. The Ontology Alignment Format [13] and the Ontology Mapping Language [28] are ongoing efforts to express ontology mappings. In a recent work [14], these two languages were merged and given a model-theoretic semantics which can be grounded to a particular logical formalism in order to be actually used to perform a mediation task. Our approach combines rule and mapping specification languages using a more practical approach than the above mentioned, exploiting standard languages, ρdf and SPARQL. We keep the ontology language expressivity low on purpose in order to retain decidability, thus providing an executable mapping specification format.

5.1 Differences with the Latest SPARQL Spec

We base our translation on the set-based semantics of [25, 26] whereas the algebra for SPARQL defined in the latest candidate recommendation [27] defines a multiset semantics. An extension of our translation towards multiset semantics is straighforward based on the observation that duplicate solution tuples for SPARQL queries can only arise from (i) projections in SELECT queries and (ii) from UNION patterns. Another slight modification of pattern semantics and translation is necessary in order to mimic the way the latest SPARQL spec deals with filters within OPTIONAL patterns that refer to variables outside the OPTIONAL part.[20] Our implementation allows to switch between the semantics defined in this paper and the fully spec-compliant version.

6 Conclusions and Further Work

In this paper we have demonstrated the use of SPARQL++ as a rule language for defining mappings between RDF vocabularies, allowing CONSTRUCT queries — extended with built-in and aggregate functions — as part of the dataset of SPARQL queries. We mainly aimed at setting the theoretical foundations for SPARQL++. Our next steps will involve to focus on scalability of our current prototype, by looking into how far evaluation of SPARQL++ queries can be optimized, for instance, by pushing query evaluation

[20]see `http://lists.w3.org/Archives/Public/public-rdf-dawg/2006OctDec/0115.html`

from our dlvhex as far as possible into more efficient SPARQL engines or possibly distributed SPARQL endpoints that cannot deal with extended datasets natively. Further, we will investigate the feasibility of supporting larger fragments of RDFS and OWL. Here, caution is in order as arbitrary combininations of OWL and SPARQL++ involve the same problems as combining rules with ontologies (see[11]) in the general case. We believe that the small fragment we started with is the right strategy in order to allow queries over networks of lightweight RDFS ontologies, connectable via expressive mappings, which we will gradually extend.

References

[1] F. Alkhateeb, J.-F. Baget, J. Euzenat. Extending SPARQL with Regular Expression Patterns. Tech. Report 6191, Inst. National de Recherche en Informatique et Automatique, May 2007.

[2] A. Borgida, L. Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics*, 1:153–184, 2003.

[3] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *The Semantic Web - ISWC 2003*, Florida, USA, 2003.

[4] W. Chen, M. Kifer, D. Warren. HiLog: A Foundation for Higher-Order Logic Programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[5] FOAF Vocabulary Specification, July 2005. `http://xmlns.com/foaf/0.1/`.

[6] J. de Bruijn, E. Franconi, S. Tessaris. Logical Reconstruction of Normative RDF. In *OWL: Experiences and Directions Workshop (OWLED-2005)*, Galway, Ireland, 2005.

[7] J. de Bruijn, S. Heymans. A Semantic Framework for Language Layering in WSML. In *First Int'l Conf. on Web Reasoning and Rule Systems (RR2007)*, Innsbruck, Austria, 2007.

[8] J. de Bruijn (ed.). Web Rule Language (WRL), 2005. W3C Member Submission.

[9] S. Decker et al. TRIPLE - an RDF Rule Language with Context and Use Cases. In *W3C Workshop on Rule Languages for Interoperability*, Washington D.C., USA, April 2005.

[10] E. Dumbill. DOAP: Description of a Project. `http://usefulinc.com/doap/`.

[11] T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, H. Tompits. Reasoning with Rules and Ontologies. In *Reasoning Web 2006*, pp. 93–127. Springer, Sept. 2006.

[12] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *International Joint Conference on Artificial Intelligence (IJCAI) 2005*, pp. 90–96, Edinburgh, UK, Aug. 2005.

[13] J. Euzenat. An API for Ontology Alignment. In *Proc. 3rd International Semantic Web Conference, Hiroshima, Japan*, pp. 698–712, 2004.

[14] J. Euzenat, F. Scharffe, A. Zimmerman. Expressive Alignment Language and Implementation. Project Deliverable D2.2.10, Knowledge Web NoE (EU-IST-2004-507482), 2007.

[15] W. Faber, N. Leone, G. Pfeifer. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. *9th European Conference on Artificial Intelligence (JELIA 2004).* Lisbon Portugal, 2004.

[16] M. Gelfond, V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[17] C. Ghidini, F. Giunchiglia. Local model semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

[18] P. Hayes. RDF Semantics. Technical Report, W3C, February 2004. W3C Recommendation.

[19] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. W3C Member Submission.

[20] R. Iannella. Representing vCard objects in RDF/XML, Feb. 2001. W3C Note.

[21] C. Kiefer, A. Bernstein, H. J. Lee, M. Klein, M. Stocker. Semantic Process Retrieval with iSPARQL. *4th European Semantic Web Conference (ESWC '07).* Innsbruck, Austria, 2007.

[22] M. Kifer, G. Lausen, J. Wu. Logical Foundations of Object-oriented and Frame-based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[23] A. Malhotra, J. Melton, N. W. (eds.). XQuery 1.0 and XPath 2.0 Functions and Operators, Jan. 2007. W3C Recommendation.

[24] S. Muñoz, J. Pérez, C. Gutierrez. Minimal Deductive Systems for RDF. *4th European Semantic Web Conference (ESWC'07)*, Innsbruck, Austria, 2007.

[25] J. Pérez, M. Arenas, C. Gutierrez. Semantics and Complexity of SPARQL. In *International Semantic Web Conference (ISWC 2006)*, pp. 30–43, 2006.

[26] A. Polleres. From SPARQL to Rules (and back). *16th World Wide Web Conference (WWW2007)*, Banff, Canada, May 2007.

[27] E. Prud'hommeaux, A. Seaborne (eds.). SPARQL Query Language for RDF, June 2007. W3C Candidate Recommendation.

[28] F. Scharffe, J. de Bruijn. A Language to specify Mappings between Ontologies. In *First Int. Conf. on Signal-Image Technology and Internet-Based Systems (IEEE SITIS'05)*, 2005.

[29] S. Schenk, S. Staab. Networked rdf graphs. Tech. Report, Univ. Koblenz, 2007. `http://www.uni-koblenz.de/~sschenk/publications/2006/ngtr.pdf`.

[30] R. Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, Vienna University of Technology, Dec. 2006.

[31] J. Ullman. *Principles of Database & Knowledge Base Systems*. Comp. Science Press, 1989.

[32] M. Völkel. RDF (Open Source) Software Vocabulary. `http://xam.de/ns/os/`.

Published in Proceedings of the 8th International Semantic Web Conference (ISWC 2009), pp. 310–327, Oct. 2007, Springer LNCS vol. 5823

Dynamic Querying of Mass-Storage RDF Data with Rule-Based Entailment Regimes*

Giovambattista Ianni[†] *Thomas Krennwallner*[‡]
Alessandra Martello[†] *Axel Polleres*[§]

[†] *Dipartimento di Matematica, Università della Calabria, I-87036 Rende (CS), Italy*
{ianni,a.martello}@mat.unical.it
[‡] *Institut für Informationssysteme 184/3, Technische Universität Wien, Austria Favoritenstraße 9-11, A-1040 Vienna, Austria*
tkren@kr.tuwien.ac.at
[§] *Digital Enterprise Research Institute, National University of Ireland, Galway*
axel.polleres@deri.org

Abstract

RDF Schema (RDFS) as a lightweight ontology language is gaining popularity and, consequently, tools for scalable RDFS inference and querying are needed. SPARQL has become recently a W3C standard for querying RDF data, but it mostly provides means for querying simple RDF graphs only, whereas querying with respect to RDFS or other entailment regimes is left outside the current specification. In this paper, we show that SPARQL faces certain unwanted ramifications when querying ontologies in conjunction with RDF datasets that comprise multiple named graphs, and we provide an extension for SPARQL that remedies these effects. Moreover, since RDFS inference has a close relationship with logic rules, we generalize our approach to select a custom ruleset for specifying inferences to be taken into account in a SPARQL query. We show that our extensions are technically feasible by providing benchmark results for RDFS querying in our prototype system GiaBATA, which uses Datalog coupled with a persistent Relational Database as a back-end for implementing SPARQL with dynamic rule-based inference. By employing different optimization techniques like magic set rewriting our system remains competitive with state-of-the-art RDFS querying systems.

*This work has been partially supported by the Italian Research Ministry (MIUR) project Interlink II04CG8AGG, the Austrian Science Fund (FWF) project P20841, by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

1 Introduction

Thanks to initiatives such as DBPedia or the Linked Open Data project,[1] a huge amount of machine-readable RDF [1] data is available, accompanying pervasive ontologies describing this data such as FOAF [2], SIOC [3], or YAGO [4].

A vast amount of Semantic Web data uses rather small and lightweight ontologies that can be dealt with rule-based RDFS and OWL reasoning [5, 6, 7], in contrast to the full power of expressive description logic reasoning. However, even if many practical use cases do not require complete reasoning on the terminological level provided by DL-reasoners, the following tasks become of utter importance. First, a Semantic Web system should be able to handle and evaluate (possibly complex) queries on large amounts of RDF instance data. Second, it should be able to take into account implicit knowledge found by ontological inferences as well as by additional custom rules involving built-ins or even nonmonotonicity. The latter features are necessary, e.g., for modeling complex mappings [8] between different RDF vocabularies. As a third point, joining the first and the second task, if we want the Semantic Web to be a solution to – as Ora Lassila formulated it – *those problems and situations that we are yet to define*,[2] we need triple stores that allow dynamic querying of different data graphs, ontologies, and (mapping) rules harvested from the Web. The notion of *dynamic* querying is in opposition to *static* querying, meaning that the same dataset, depending on context, reference ontology and entailment regime, might give different answers to the same query. Indeed, there are many situations in which the dataset at hand and its supporting class hierarchy cannot be assumed to be known upfront: think of distributed querying of remotely exported RDF data.

Concerning the first point, traditional RDF processors like Jena (using the default configuration) are designed for handling large RDF graphs in memory, thus reaching their limits very early when dealing with large graphs retrieved from the Web. Current RDF Stores, such as YARS [9], Sesame, Jena TDB, ThreeStore, AllegroGraph, or OpenLink Virtuoso[3] provide roughly the same functionality as traditional relational database systems do for relational data. They offer query facilities and allow to import large amounts of RDF data into their persistent storage, and typically support SPARQL [10], the W3C standard RDF query language. SPARQL has the same expressive power as non-recursive Datalog [11, 12] and includes a set of built-in predicates in so called **`filter`** expressions.

However, as for the second and third point, current RDF stores only offer limited support. OWL or RDF(S) inference, let alone custom rules, are typically fixed in combination with SPARQL querying (cf. Section 2). Usually, dynamically assigning different ontologies or rulesets to data for querying is neither supported by the SPARQL specification nor by existing systems. Use cases for such dynamic querying involve, e.g., querying data with different versions of ontologies or queries over data expressed in related ontologies adding custom mappings (using rules or "bridging" ontologies).

To this end, we propose an extension to SPARQL which caters for knowledge-intensive applications on top of Semantic Web data, combining SPARQL querying with

[1]`http://dbpedia.org/` and `http://linkeddata.org/`

[2]`http://www.lassila.org/publications/2006/SCAI-2006-keynote.pdf`

[3]See `http://openrdf.org/`, `http://jena.hpl.hp.com/wiki/TDB/`, `http://threestore.sf.net/`, `http://agraph.franz.com/allegrograph/`, `http://openlinksw.com/virtuoso/`, respectively.

dynamic, rule-based inference. In this framework, we overcome some of the above mentioned limitations of SPARQL and existing RDF stores. Moreover, our approach is easily extensible by allowing features such as aggregates and arbitrary built-in predicates to SPARQL (see [8, 14]) as well as the addition of custom inference and mapping rules. The contributions of our paper are summarized as follows:

• We introduce two additional language constructs to the normative SPARQL language. First, the directive **`using ontology`** for dynamically coupling a dataset with an arbitrary RDFS ontology, and second *extended dataset clauses*, which allow to specify datasets with named graphs in a flexible way. The **`using ruleset`** directive can be exploited for adding to the query at hand proper rulesets which might used for a variety of applications such as encoding mappings between entities, or encoding custom entailment rules, such as RDFS or different rule-based OWL fragments.

• We present the GiaBATA system [15], which demonstrates how the above extensions can be implemented on a middle-ware layer translating SPARQL to Datalog and SQL. Namely, the system is based on known translations of SPARQL to Datalog rules. Arbitrary, possibly recursive rules can be added flexibly to model arbitrary ontological inference regimes, vocabulary mappings, or alike. The resulting program is compiled to SQL where possible, such that only the recursive parts are evaluated by a native Datalog implementation. This hybrid approach allows to benefit from efficient algorithms of deductive database systems for custom rule evaluation, and native features such as query plan optimization techniques or rich built-in functions (which are for instance needed to implement complex **`filter`** expressions in SPARQL) of common database systems.

• We compare our GiaBATA prototype to well-known RDF(S) systems and provide experimental results for the LUBM [16] benchmark. Our approach proves to be competitive on both RDF and dynamic RDFS querying without the need to pre-materialize inferences.

In the remainder of this paper we first introduce SPARQL along with RDF(S) and partial OWL inference by means of some motivating example queries which existing systems partially cannot deal in a reasonably manner in Section 2. Section 3 sketches how the SPARQL language can be enhanced with custom ruleset specifications and arbitrary graph merging specifications. We then briefly introduce our approach to translate SPARQL rules to Datalog in Section 4, and how this is applied to a persistent storage system. We evaluate our approach with respect to existing RDF stores in Section 5, and then conclusions are drawn in Section 6.

2 SPARQL and some Motivating Examples

Similar in spirit to structured query languages like SQL, which allow to extract, combine and filter data from relational database tables, SPARQL allows to extract, combine and filter data from RDF graphs. The semantics and implementation of SPARQL involves, compared to SQL, several peculiarities, which we do not focus on in this paper, cf. [10, 18, 11, 19] for details. Instead, let us just start right away with some illustrating example motivating our proposed extensions of SPARQL; we assume two data graphs describing data about our well-known friends Bob and Alice shown in Fig. 1(b)+(c).

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/>.
@prefix rel:  <http://purl.org/vocab/relationship/>.
...
rel:friendOf rdfs:subPropertyOf foaf:knows.
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:homepage rdf:type owl:inverseFunctionalProperty.
...
```

(a) Graph G_M (`<http://example.org/myOnt.rdfs>`), a combination of the FOAF&Relationship ontologies.

```
<http://bob.org#me> foaf:name "Bob";
a foaf:Person;
   foaf:homepage
<http://bob.org/home.html>;
   rel:friendOf [ foaf:name "Alice";
                  rdfs:seeAlso
<http://alice.org> ].
```

(b) Graph G_B (`<http://bob.org>`)

```
<http://alice.org#me> foaf:name "Alice";
a foaf:Person;
rel:friendOf [ foaf:name "Charles" ],
             [ foaf:name "Bob";
               foaf:homepage
<http://bob.org/home.html> ].
```

(c) Graph G_A (`<http://alice.org>`)

Figure 1: An ontology and two data graphs

Both graphs refer to terms in a combined ontology defining the FOAF and Relationship[4] vocabularies, see Fig. 1(a) for an excerpt.

On this data the SPARQL query (1) intends to extract names of persons mentioned in those graphs that belong to friends of Bob. We assume that, by means of `rdfs:seeAlso` statements, Bob provides links to the graphs associated to the persons he is friend with.

```
select ?N from <http://example.org/myOnt.rdfs>
          from <http://bob.org>
          from named <http://alice.org>                              (1)
where { <http://bob.org#me> foaf:knows ?X . ?X rdfs:seeAlso ?G .
        graph ?G { ?P rdf:type foaf:Person; foaf:name ?N } }
```

Here, the **`from`** and **`from named`** clauses specify an RDF dataset. In general, the *dataset* $DS = (G, N)$ of a SPARQL query is defined by (i) a *default graph* G obtained by the RDF merge [20] of all graphs mentioned in **`from`** clauses, and (ii) a set $N = \{(u_1, G_1), \ldots, (u_k, G_k)\}$ of *named graphs*, where each pair (u_i, G_i) consists of an IRI u_i, given in a **`from named`** clause, paired with its corresponding graph G_i. For instance, the dataset of query (1) would be $DS_1 = (\, G_M \uplus G_B, \{($`<http://alice.org>`$, G_A)\})$, where $\uplus$ denotes merging of graphs according to the normative specifications.

Now, let us have a look at the answers to query (1). Answers to SPARQL **`select`** queries are defined in terms of multisets of partial variable substitutions. In fact the answer to query (1) is empty when – as typical for current SPARQL engines – only simple RDF entailment is taken into account, and query answering then boils down to simple graph matching. Since neither of the graphs in the default graph contain any triple matching the pattern `<http://bob.org#me> foaf:knows ?X` in the **`where`** clause, the result of (1) is empty. When taking subproperty inference by the statements

[4]`http://vocab.org/relationship/`

of the ontology in G_M into account, however, one would expect to obtain three substitutions for the variable ?N: {?N/"Alice", ?N/"Bob", ?N/"Charles"}. We will explain in the following why this is not the case in standard SPARQL.

In order to obtain the expected answer, firstly SPARQL's basic graph pattern matching needs to be extended, see [10, Section 12.6]. In theory, this means that the graph patterns in the **where** clause needs to be matched against an enlarged version of the original graphs in the dataset (which we will call the *deductive closure* $Cl(\cdot)$) of a given entailment regime. Generic extensions for SPARQL to entailment regimes other than simple RDF entailment are still an open research problem,[5] due to various problems: (i) for (non-simple) RDF entailment regimes, such as full RDFS entailment, $Cl(G)$ is infinite, and thus SPARQL queries over an empty graph G might already have infinite answers, and (ii) it is not yet clear which should be the intuitive answers to queries over inconsistent graphs, e.g. in OWL entailment, etc. In fact, SPARQL restricts extensions of basic graph pattern matching to retain finite answers. Not surprisingly, many existing implementations implement finite approximations of higher entailment regimes such as RDFS and OWL [6, 5, 21]. E.g., the RDF Semantics document [20] contains an informative set of entailment rules, a subset of which (such as the one presented in Section 3.2 below) is implemented by most available RDF stores. These rule-based approximations, which we focus on in this paper, are typically expressible by means of Datalog-style rules. These latter model how to infer a finite closure of a given RDF graph that covers sound but not necessarily complete RDF(S) and OWL inferences. It is worth noting that Rule-based entailment can be implemented in different ways: rules could be either dynamically evaluated upon query time, or the closure wrt. ruleset $\mathcal{R}$, $Cl_{\mathcal{R}}(G)$, could be materialized when graph G is loaded into a store. Materialization of inferred triples at loading time allows faster query responses, yet it has drawbacks: it is time and space expensive and it has to be performed once and statically. In this setting, it must be decided upfront

(a) which ontology should be taken into account for which data graph, and

(b) to which graph(s) the inferred triples "belong", which particularly complicates the querying of named graphs.

As for exemplifying (a), assume that a user agent wants to issue another query on graph G_B with only the FOAF ontology in mind, since she does not trust the Relationship ontology. In the realm of FOAF alone, `rel:friendOf` has nothing to deal with `foaf:knows`. However, when materializing all inferences upon loading G_M and G_B into the store, `bob:me foaf:knows _:a` would be inferred from $G_M \uplus G_B$ and would contribute to such a different query. Current RDF stores prevent to dynamically parameterize inference with an ontology of choice at query time, since indeed typically all inferences are computed upon loading time *once and for all*.

As for (b), queries upon datasets including named graphs are even more problematic. Query (1) uses G_B in order to find the IRI identifiers for persons that Bob knows by following `rdfs:seeAlso` links and looks for persons and their names in the *named* RDF graphs found at these links. Even if rule-based inference was supported, the answer to query (1) over dataset DS_1 is just {?N/"Alice"}, as "Alice" is the only (explicitly) asserted `foaf:Person` in G_A. Subproperty, domain and range inferences over the G_M ontology do not propagate to G_A, since G_M is normatively prescribed to

[5]For details, cf. `http://www.polleres.net/sparqltutorial/`, Unit 5b.

be merged into the default graph, but not into the named graph. Thus, there is no way to infer that `"Bob"` and `"Charles"` are actually names of `foaf:Person`s within the named graph G_A. Indeed, SPARQL does not allow to merge, on demand, graphs into the named graphs, thus there is no way of combining G_M with the named graph G_A.

To remedy these deficiencies, we suggest an extension of the SPARQL syntax, in order to allow the specification of datasets more flexibly: it is possible to group graphs to be merged in parentheses in **`from`** and **`from named`** clauses. The modified query, obtaining a dataset $DS_2 = (\, G_M \uplus G_B, \{(\texttt{http://alice.org}, G_M \uplus G_A)\})$ looks as follows:

```
select ?N
  from (<http://example.org/myOnt.rdfs> <http://bob.org/>)
  from named <http://alice.org>
     (<http://example.org/myOnt.rdfs> <http://alice.org/>)                (2)
where { bob:me foaf:knows ?X . ?X rdfs:seeAlso ?G .
        graph ?G { ?X foaf:name ?N . ?X a foaf:Person . } }
```

For ontologies which should apply to the whole query, i.e., graphs to be merged into the default graph as well as any specified named graph, we suggest a more convenient shortcut notation by adding the keyword **`using ontology`** in the SPARQL syntax:

```
select ?N
  using ontology <http://example.org/myOnt.rdfs>
  from <http://bob.org/>
  from named <http://alice.org/>                                          (3)
where { bob:me foaf:knows ?X . ?X foaf:seeAlso ?G .
        graph ?G { ?X foaf:name ?N . ?X a foaf:Person. } }
```

Hence, the **`using ontology`** construct allows for coupling the entire given dataset with the terminological knowledge in the `myOnt` data schema. As our investigation of currently available RDF stores (see Section 5) shows, none of these systems easily allow to merge ontologies into named graphs or to dynamically specify the dataset of choice.

In addition to parameterizing queries with ontologies in the dataset clauses, we also allow to parameterize the ruleset which models the entailment regime at hand. Per default, our framework supports a standard ruleset that “emulates” (a finite subset of) the RDFS semantics. This standard ruleset is outlined in Section 3 below. Alternatively, different rule-based entailment regimes, e.g., rulesets covering parts of the OWL semantics á la ter Horst [5], de Bruijn [22, Section 9.3], OWL2 RL [17] or other custom rulesets can be referenced with the **`using ruleset`** keyword. For instance, the following query returns the solution $\{?\mathtt{X}/$`<http://alice.org#me>`, $?\mathtt{Y}/$`<http://bob.org#me>`$\}$, by doing equality reasoning over inverse functional properties such as `foaf:homepage` when the FOAF ontology is being considered:

```
select ?X ?Y
 using ontology <http://example.org/myOnt.rdfs>
 using ruleset rdfs
 using ruleset <http://www.example.com/owl-horst>                         (4)
 from <http://bob.org/>
 from <http://alice.org/>
where { ?X foaf:knows ?Y }
```

Query (4) uses the built-in RDFS rules for the usual subproperty inference, plus a ruleset implementing ter Horst’s inference rules, which might be available at URL `http://www.example.com/owl-horst`. This ruleset contains the following additional rules, that will "equate" the blank node used in G_A for "Bob" with `<http://bob.org#me>`:[6]

```
?P rdf:type owl:iFP . ?S1 ?P ?O . ?S2 ?P ?O .    → ?S1 owl:sameAs ?S2.
?X owl:sameAs ?Y                                 → ?Y owl:sameAs ?X.
?X ?P ?O . ?X owl:sameAs ?Y                      → ?Y ?P ?O.            (5)
?S ?X ?O . ?X owl:sameAs ?Y                      → ?S ?Y ?O.
?S ?P ?X . ?X owl:sameAs ?Y                      → ?S ?P ?Y.
```

3 A Framework for Using Ontologies and Rules in SPARQL

In the following, we will provide a formal framework for the SPARQL extensions outlined above. In a sense, the notion of *dynamic querying* is formalized in terms of the dependence of BGP pattern answers $[\![P]\!]^{\mathcal{O},\mathcal{R}}$ from a variable ontology $\mathcal{O}$ and ruleset $\mathcal{R}$. For our exposition, we rely on well-known definitions of RDF datasets and SPARQL. Due to space limitations, we restrict ourselves to the bare minimum and just highlight some standard notation used in this paper.

Preliminaries. Let I, B, and L denote pairwise disjoint infinite sets of IRIs, blank nodes, and RDF literals, respectively. A *term* is an element from $I \cup B \cup L$. An *RDF graph* G (or simply *graph*) is defined as a set of *triples* from $I \cup B \times I \cup B \times I \cup B \cup L$ (cf. [18, 12]); by $blank(G)$ we denote the set of blank nodes of G.[7]

A *blank node renaming* θ is a mapping $I \cup B \cup L \rightarrow I \cup B \cup L$. We denote by $t\theta$ the application of θ to a term t. If $t \in I \cup L$ then $t\theta = t$, and if $t \in B$ then $t\theta \in B$. If (s, p, o) is a triple then $(s, p, o)\theta$ is the triple $(s\theta, p\theta, o\theta)$. Given a graph G, we denote by $G\theta$ the set of all triples $\{t\theta \mid t \in G\}$. Let G and H be graphs. Let θ^G_H be an arbitrary blank node renaming such that $blank(G) \cap blank(H\theta^G_H) = \emptyset$. The *merge of G by H*, denoted $G \uplus H$, is defined as $G \cup H\theta^G_H$.

An *RDF dataset* $D = (G_0, N)$ is a pair consisting of exactly one unnamed graph, the so-called default graph G_0, and a set $N = \{\langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$ of named graphs, coupled with their identifying URIs. The following conditions hold: (i) each G_i ($0 \leq i \leq n$) is a graph, (ii) each u_j ($1 \leq j \leq n$) is from I, and (iii) for all $i \neq j$, $\langle u_i, G_i\rangle, \langle u_j, G_j\rangle \in N$ implies $u_i \neq u_j$ and $blank(G_i) \cap blank(G_j) = \emptyset$.

The syntax and semantics of SPARQL can now be defined as usual, cf. [10, 18, 12] for details. For the sake of this paper, we restrict ourselves to **`select`** queries as shown in the example queries (1)–(4) and just provide an overview of the necessary concepts. A query in SPARQL can be viewed as a tuple $Q = (V, D, P)$, where V is the set of variables mentioned in the **`select`** clause, D is an RDF dataset, defined by means of **`from`** and **`from named`** clauses, and P is a *graph pattern*, defined in the **`where`** clause.

Graph patterns are in the simplest case sets of RDF triples (s, p, o), where terms and variables from an infinite set of variables *Var* are allowed, also called *basic graph patterns (BGP)*. More complex graph patterns can be defined recursively, i.e., if P_1 and P_2 are graph patterns, $g \in I \cup Var$ and R is a filter expression, then P_1 **`optional`** P_2, P_1 **`union`** P_2, P_1 **`filter`** R, and **`graph`** $g\, P_1$ are graph patterns.

Graph pattern matching. Queries are evaluated by matching graph patterns against graphs in the dataset. In order to determine a query's solution, in the simplest case BGPs are matched against the *active graph* of the query, which is one particular graph in the dataset, identified as shown next.

[6] We use `owl:iFP` as shortcut for `owl:inverseFunctionalProperty`.

[7] Note that we allow *generalized RDF graphs* that may have blank nodes in property position.

Solutions of BGP matching consist of multisets of bindings for the variables mentioned in the pattern to terms in the active graph. Partial solutions of each subpattern are joined according to an algebra defining the **`optional`**, **`union`** and **`filter`** operators, cf. [10, 18, 12]. For what we are concerned with here, the most interesting operator though is the **`graph`** operator, since it changes the active graph. That is, the active graph is the default graph G_0 for any basic graph pattern not occurring within a **`graph`** sub pattern. However, in a subpattern **`graph`** $g\,P_1$, the pattern P_1 is matched against the named graph identified by g, if $g \in I$, and against any named graph u_i, if $g \in \mathit{Var}$, where the binding u_i is returned for variable g. According to [12], for a RDF dataset D and active graph G, we define $[\![P]\!]^D_G$ as the multiset of tuples constituting the *answer* to the graph pattern P. The solutions of a query $Q = (V, D, P)$ is the projection of $[\![P]\!]^D_G$ to the variables in V only.

3.1 SPARQL with Extended Datasets

What is important to note now is that, by the way how datasets are syntactically defined in SPARQL, the default graph G_0 can be obtained from merging a group of different source graphs, specified via several **`from`** clauses – as shown, e.g., in query (1) – whereas in each **`from named`** clause a single, separated, named graph is added to the dataset. That is, **`graph`** patterns will always be matched against a separate graph only. To generalize this towards dynamic construction of groups of merged named graphs, we introduce the notion of an *extended dataset*, which can be specified by enlarging the syntax of SPARQL with two additional dataset clauses:

- For $i, i_1, \ldots, i_m$ distinct IRIs ($m \geq 1$), the statement "**`from named`** $i(i_1 \ldots i_m)$" is called *extended dataset clause*. Intuitively, $i_1 \ldots i_m$ constitute a group of graphs to be merged: the merged graph is given i as identifying IRI.

- For $o \in I$ we call the statement "**`using ontology`** o" an *ontological dataset clause*. Intuitively, o stands for a graph that will merged with all graphs in a given query.

Extended RDF datasets are thus defined as follows. A *graph collection* $\mathcal{G}$ is a set of RDF graphs. An *extended RDF dataset* $\mathcal{D}$ is a pair $(\mathcal{G}_0, \{\langle u_1, \mathcal{G}_1\rangle, \ldots, \langle u_n, \mathcal{G}_n\rangle\})$ satisfying the following conditions: (i) each $\mathcal{G}_i$ is a nonempty graph collection (note that $\{\emptyset\}$ is a valid nonempty graph collection), (ii) each u_j is from I, and (iii) for all $i \neq j$, $\langle u_i, \mathcal{G}_i\rangle, \langle u_j, \mathcal{G}_j\rangle \in D$ implies $u_i \neq u_j$ and for $G \in \mathcal{G}_i$ and $H \in \mathcal{G}_j$, $\mathit{blank}(G) \cap \mathit{blank}(H) = \emptyset$. We denote $\mathcal{G}_0$ as $dg(\mathcal{D})$, the *default graph collection* of $\mathcal{D}$.

Let $\mathcal{D}$ and $\mathcal{O}$ be an extended dataset and a graph collection, resp. The ordinary RDF dataset obtained from $\mathcal{D}$ and $\mathcal{O}$, denoted $D(\mathcal{D}, \mathcal{O})$, is defined as

$$\left(\biguplus_{g \in dg(\mathcal{D})} g \uplus \biguplus_{o \in \mathcal{O}} o, \left\{ \langle u, \biguplus_{g \in \mathcal{G}} g \uplus \biguplus_{o \in \mathcal{O}} o \rangle \mid \langle u, \mathcal{G} \rangle \in \mathcal{D} \right\} \right) .$$

We can now define the semantics of extended and ontological dataset clauses as follows. Let F be a set of ordinary and extended dataset clauses, and O be a set of ontological dataset clauses. Let $graph(g)$ be the graph associated to the IRI g: the extended RDF dataset obtained from F, denoted $edataset(F)$, is composed of:

(1) $\mathcal{G}_0 = \{graph(g) \mid$ "**`from`** g" $\in F\}$. If there is no **`from`** clause, then $\mathcal{G}_0 = \emptyset$.

(2) A named graph collection $\langle u, \{graph(u)\}\rangle$ for each "**`from named`** u" in F.

(3) A named graph collection $\langle i, \{graph(i_1), \ldots, graph(i_m)\}\rangle$ for each "**`from named`** $i(i_1 \ldots i_m)$" in F.

The graph collection obtained from O, denoted $ocollection(O)$, is the set $\{graph(o) \mid$ "**`using ontology`** o" $\in O\}$. The ordinary dataset of O and F, denoted $dataset(F, O)$, is the set $D(edataset(F), ocollection(O))$.

Let $\mathcal{D}$ and $\mathcal{O}$ be as above. The *evaluation* of a graph pattern P over $\mathcal{D}$ and $\mathcal{O}$ having active graph collection $\mathcal{G}$, denoted $[\![P]\!]_{\mathcal{G}}^{\mathcal{D},\mathcal{O}}$, is the evaluation of P over $D(\mathcal{D}, \mathcal{O})$ having active graph $G = \biguplus_{g \in \mathcal{G}} g$, that is, $[\![P]\!]_{\mathcal{G}}^{\mathcal{D},\mathcal{O}} = [\![P]\!]_{G}^{D(\mathcal{D},\mathcal{O})}$.

Note that the semantics of extended datasets is defined in terms of ordinary RDF datasets. This allows to define the semantics of SPARQL with extended and ontological dataset clauses by means of the standard SPARQL semantics. Also note that our extension is conservative, i.e., the semantics coincides with the standard SPARQL semantics whenever no ontological clauses and extended dataset clauses are specified.

3.2 SPARQL with Arbitrary Rulesets

Extended dataset clauses give the possibility of merging arbitrary ontologies into any graph in the dataset. The second extension herein presented enables the possibility of dynamically deploying and specifying rule-based entailments regimes on a per query basis. To this end, we define a generic $\mathcal{R}$-entailment, that is RDF entailment associated to a parametric ruleset $\mathcal{R}$ which is taken into account when evaluating queries. For each such $\mathcal{R}$-entailment regime we straightforwardly extend BGP matching, in accordance with the conditions for such extensions as defined in [10, Section 12.6].

We define an *RDF inference rule* r as the pair $(\mathcal{A}nte, \mathcal{C}on)$, where the *antecedent* $\mathcal{A}nte$ and the *consequent* $\mathcal{C}on$ are basic graph patterns such that $\mathcal{V}(\mathcal{C}on)$ and $\mathcal{V}(\mathcal{A}nte)$ are non-empty, $\mathcal{V}(\mathcal{C}on) \subseteq \mathcal{V}(\mathcal{A}nte)$ and $\mathcal{C}on$ does not contain blank nodes.[8] As in Example (5) above, we typically write RDF inference rules as

$$\mathcal{A}nte \rightarrow \mathcal{C}on \ . \tag{6}$$

We call sets of inference rules *RDF inference rulesets*, or *rulesets* for short.

Rule Application and Closure. We define *RDF rule application* in terms of the immediate consequences of a rule r or a ruleset $\mathcal{R}$ on a graph G. Given a BGP P, we denote as $\mu(P)$ a pattern obtained by substituting variables in P with elements of $I \cup B \cup L$. Let r be a rule of the form (6) and G be a set of RDF triples, then:

$$T_r(G) = \{\mu(\mathcal{C}on) \mid \exists\mu \text{ such that } \mu(\mathcal{A}nte) \subseteq G\}.$$

Accordingly, let $T_{\mathcal{R}}(G) = \bigcup_{r \in \mathcal{R}} T_r(G)$. Also, let $G_0 = G$ and $G_{i+1} = G_i \cup T_{\mathcal{R}}(G_i)$ for $i \geq 0$. It can be easily shown that there exists the smallest n such that $G_{n+1} = G_n$; we call then $Cl_{\mathcal{R}}(G) = G_n$ the *closure* of G with respect to ruleset $\mathcal{R}$.

We can now further define *$\mathcal{R}$-entailment* between two graphs G_1 and G_2, written $G_1 \models_{\mathcal{R}} G_2$, as $Cl_{\mathcal{R}}(G_1) \models G_2$. Obviously for any finite graph G, $Cl_{\mathcal{R}}(G)$ is finite.

[8]Unlike some other rule languages for RDF, the most prominent of which being CONSTRUCT statements in SPARQL itself, we forbid blank nodes; i.e., existential variables in rule consequents which require the "invention" of new blank nodes, typically causing termination issues.

In order to define the semantics of a SPARQL query wrt. $\mathcal{R}$*-entailment* we now extend graph pattern matching in $[\![P]\!]^D_G$ towards respecting $\mathcal{R}$.

Definition 23 (extended basic graph pattern matching for $\mathcal{R}$-entailment) *Let D be a dataset and G be an active graph. The* solution of a BGP P wrt. $\mathcal{R}$-entailment*, denoted $[\![P]\!]^{D,\mathcal{R}}_G$, is $[\![P]\!]^D_{Cl_{\mathcal{R}}(G)}$.*

The solution $[\![P]\!]^{D,\mathcal{R}}_G$ naturally extends to more complex patterns according to the SPARQL algebra. In the following we will assume that $[\![P]\!]^{D,\mathcal{R}}_G$ is used for graph pattern matching. Our extension of basic graph pattern matching is in accordance with the conditions for extending BGP matching in [10, Section 12.6]. Basically, these conditions say that any extension needs to guarantee finiteness of the answers, and defines some conditions about a "*scoping graph*." Intuitively, for our extension, the scoping graph is just equivalent to $Cl_{\mathcal{R}}(G)$. We refer to [10, Section 12.6] for the details.

To account for this generic SPARQL BGP matching extension parameterized by an RDF inference ruleset $\mathcal{R}_Q$ per SPARQL query Q, we introduce another novel language construct for SPARQL:

- For $r \in I$ we call "**`using ruleset`** r" a *ruleset clause*.

Analogously to IRIs denoting graphs, we now assume that an IRI $r \in I$ may not only refer to graphs but also to rulesets, and denote the corresponding ruleset by $ruleset(r)$. Each query Q may contain zero or more ruleset clauses, and we define the query ruleset $\mathcal{R}_Q = \bigcup_{r \in R} ruleset(r)$, where R is the set of all ruleset clauses in Q.

The definitions of solutions of a query and the evaluation of a pattern in this query on active graph G is now defined just as above, with the only difference that answer to a pattern P are given by $[\![P]\!]^{D,\mathcal{R}_Q}_G$.

We observe that whenever $\mathcal{R} = \emptyset$, then $\mathcal{R}$-entailment boils down to simple RDF entailment. Thus, a query without ruleset clauses will just be evaluated using standard BGP matching. In general, our extension preserve full backward compatibility.

Proposition 25 *For $\mathcal{R} = \emptyset$ and RDF graph G, $[\![P]\!]^{D,\mathcal{R}}_G = [\![P]\!]^D_G$.*

Analogously, one might use $\mathcal{R}$*-entailment* as the basis for RDFS entailment as follows. We consider here the ρDF fragment of RDFS entailment [6]. Let $\mathcal{R}_{RDFS}$ denote the ruleset corresponding to the minimal set of entailment rules (2)–(4) from [6]:

```
?P rdfs:subPropertyOf ?Q . ?Q rdfs:subPropertyOf ?R .   → ?P rdfs:subPropertyOf ?R.
?P rdfs:subPropertyOf ?Q . ?S ?P ?O .                   → ?S ?Q ?O.
?C rdfs:subClassOf ?D . ?D rdfs:subClassOf ?E .         → ?C rdfs:subClassOf ?E.
?C rdfs:subClassOf ?D . ?S rdf:type ?C .                → ?S rdf:type ?D.
?P rdfs:domain ?C . ?S ?P ?O .                          → ?S rdf:type ?C.
?P rdfs:range ?C . ?S ?P ?O .                           → ?O rdf:type ?C.
```

Since obviously $G \models_{RDFS} Cl_{\mathcal{R}_{RDFS}}(\mathcal{G})$ and hence $Cl_{\mathcal{R}_{RDFS}}(\mathcal{G})$ may be viewed as a finite approximation of RDFS-entailment, we can obtain a reasonable definition for defining a BGP matching extension for RDFS by simply defining $[\![P]\!]^{D,RDFS}_G = [\![P]\!]^{D,\mathcal{R}_{RDFS}}_G$. We allow the special ruleset clause **`using ruleset`** `rdfs` to conveniently refer to this particular ruleset. Other rulesets may be published under a Web dereferenceable URI, e.g., using an appropriate RIF [23] syntax.

Note, eventually, that our rulesets consist of positive rules, and as such enjoy a natural monotonicity property.

Proposition 26 *For rulesets $\mathcal{R}$ and $\mathcal{R}'$, such that $\mathcal{R} \subseteq \mathcal{R}'$, and graph G_1 and G_2, if $G_1 \models_{\mathcal{R}} G_2$ then $G_1 \models_{\mathcal{R}'} G_2$.*

Entailment regimes modeled using rulesets can thus be enlarged without retracting former inferences. This for instance would allow to introduce tighter RDFS-entailment approximations by extending $\mathcal{R}_{RDFS}$ with further axioms, yet preserving inferred triples.

4 Translating SPARQL into Datalog and SQL

Our extensions have been implemented by reducing both queries, datasets and rulesets to a common ground which allows arbitrary interoperability between the three realms. This common ground is Datalog, wherein rulesets naturally fit and SPARQL queries can be reduced to. Subsequently, the resulting combined Datalog programs can be evaluated over an efficient SQL interface to an underlying relational DBMS that works as triple store.

From SPARQL to Datalog. A SPARQL query Q is transformed into a corresponding Datalog program D_Q. The principle is to break Q down to a series of Datalog rules, whose body is a conjunction of atoms encoding a graph pattern. D_Q is mostly a plain Datalog program in dlvhex [24] input format, i.e. Datalog with *external predicates* in the dlvhex language. These are explained along with a full account of the translation in [11, 19]. Main challenges in the transformation from SPARQL to Datalog are (i) faithful treatment of the semantics of joins over possibly unbound variables [11], (ii) the multiset semantics of SPARQL [19], and also (iii) the necessity of Skolemization of blank nodes in **construct** queries [8]. Treatment of **optional** statements is carried out by means of an appropriate encoding which exploits negation as failure. Special external predicates of dlvhex are used for supporting some features of the SPARQL language: in particular, importing RDF data is achieved using the external $\&rdf$ predicate, which can be seen as a built-in referring to external data. Moreover, SPARQL **filter** expressions are implemented using the dlvhex external $\&eval$ predicate in D_Q.

Let us illustrate this transformation step by an example: the following query A asking for persons who are not named “Alice” and optionally their email addresses

```
select * from <http://alice.org/>
 where { ?X a foaf:Person. ?X foaf:name ?N.                       (7)
         filter ( ?N != "Alice") optional { ?X foaf:mbox ?M } }
```

is translated to the program D_A as follows:

```
(r1) "triple"(S,P,0,default)     :- &rdf[ "alice.org" ](S,P,0).
(r2) answer1(X_N,X_X,default)    :- "triple"(X_X,"rdf:type","foaf:Person",default),
                                    "triple"(X_X,"foaf:name",X_N,default),
                                    &eval[" ?N != 'Alice' ","N", X_N ](true).
(r3) answer2(X_M,X_X,default)    :- "triple"(X_X,"foaf:mbox",X_M,default).
(r4) answer_b_join_1(X_M,X_N,X_X,default)  :- answer1(X_N,X_X,default),
                                              answer2(X_M,X_X,default).
(r5) answer_b_join_1(null,X_N,X_X,default) :- answer1(X_N,X_X,default),
                                              not answer2_prime(X_X,default).
(r6) answer2_prime(X_X,default) :- answer1(X_N,X_X,default),
                                   answer2(X_M,X_X,default).
(r7) answer(X_M,X_N,X_X)         :- answer_b_join1(X_M,X_N,X_X,default).
```

where the first rule `(r1)` computes the predicate `"triple"` taking values from the built-in predicate *&rdf*. This latter is generally used to import RDF statements from the specified URI. The following rules `(r2)` and `(r3)` compute the solutions for the filtered basic graph patterns { `?X a foaf:Person. ?X foaf:name ?N.` **`filter`** `(?N != "Alice")` } and { `?X foaf:mbox ?M`}. In particular, note here that the evaluation of **`filter`** expressions is "outsourced" to the built-in predicate *&eval*, which takes a filter expression and an encoding of variable bindings as arguments, and returns the evaluation value (`true`, `false` or `error`, following the SPARQL semantics). In order to emulate SPARQL's **`optional`** patterns a combination of join and set difference operation is used, which is established by rules `(r4)-(r6)`. Set difference is simulated by using both *null* values and *negation as failure*. According to the semantics of SPARQL, one has to particularly take care of variables which are joined and possibly unbound (i.e., set to the `null` value) in the course of this translation for the general case. Finally, the dedicated predicate *answer* in rule `(r7)` collects the answer substitutions for Q. D_Q might then be merged with additional rulesets whenever Q contains **`using ruleset`** clauses.

From Datalog to SQL. For this step we rely on the system DLV^{DB} [25] that implements Datalog under stable model semantics on top of a DBMS of choice. DLV^{DB} is able to translate Datalog programs in a corresponding SQL query plan to be issued to the underlying DBMS. RDF Datasets are simply stored in a database D, but the native dlvhex $\&rdf$ and $\&eval$ predicates in D_Q cannot be processed by DLV^{DB}directly over D. So, D_Q needs to be post-processed before it can be converted into suitable SQL statements.

Rule `(r1)` corresponds to loading persistent data into D, instead of loading triples via the $\&rdf$ built-in predicate. In practice, the predicate `"triple"` occurring in program D_A is directly associated to a database table TRIPLE in D. This operation is done off-line by a loader module which populates the TRIPLE table accordingly, while `(r1)` is removed from the program. The $\&eval$ predicate calls are recursively broken down into `WHERE` conditions in SQL statements, as sketched below when we discuss the implementation of **`filter`** statements.

After post-processing, we obtain a program D'_Q, which DLV^{DB}allows to be executed on a DBMS by translating it to corresponding SQL statements. D'_Q is coupled with a mapping file which defines the correspondences between predicate names appearing in D'_Q and corresponding table and view names stored in the DBMS D.

For instance, the rule `(r4)` of D_A, results in the following SQL statement issued to the RDBMS by DLV^{DB}:

```
INSERT INTO answer_b_join_1
 SELECT DISTINCT answer2_p2.a1, answer1_p1.a1, answer1_p1.a2, 'default'
 FROM answer1 answer1_p1, answer2 answer2_p2
 WHERE (answer1_p1.a2=answer2_p2.a2)
 AND (answer1_p1.a3='default')
 AND (answer2_p2.a3='default')
 EXCEPT (SELECT * FROM answer_b_join_1)
```

Whenever possible, the predicates for computing intermediate results such as `answer1`, `answer2`, `answer_b_join_1`, ..., are mapped to SQL views rather than materialized tables, enabling dynamic evaluation of predicate contents on the DBMS side.[9]

[9]For instance, recursive predicates require to be associated with permanent tables, while remaining predicates are normally associated to views.

Schema rewriting. Our system allows for customizing schemes which triples are stored in. It is known and debated [26] that in choosing the data scheme of D several aspects have to be considered, which affect performance and scalability when handling large-scale RDF data. A widely adopted solution is to exploit a single table storing quadruples of form (s, p, o, c) where s, p, o and c are, respectively, the triple subject, predicate, object and context the triple belongs to. This straightforward representation is easily improved [27] by avoiding to store explicitly string values referring to URIs and literals. Instead, such values are replaced with a corresponding hash value.

Other approaches suggest alternative data structures, e.g., property tables [27, 26]. These aim at denormalizing RDF graphs by storing them in a flattened representation, trying to encode triples according to the hidden "schema" of RDF data. Similarly to a traditional relational schema, in this approach D contains a table per each known property name (and often also per class, splitting up the `rdf:type` table).

Our system gives sufficient flexibility in order to program different storage schemes: while on higher levels of abstraction data are accessible via the 4-ary *triple* predicate, a schema rewriter module is introduced in order to match D'_Q to the current database scheme. This module currently adapts D'_Q by replacing constant IRIs and literals with their corresponding hash value, and introducing further rules which translate answers, converting hash values back to their original string representation.

Magic sets. Notably, DLV^{DB} can post-process D'_Q using the magic sets technique, an optimization method well-known in the database field [28]. The optimized program mD'_Q tailors the data to be queried to an extent significantly smaller than the original D'_Q. The application of magic sets allows, e.g., to apply entailment rules $\mathcal{R}_{RDFS}$ only on triples which might affect the answer to Q, preventing thus the full computation and/or materialization of inferred data.

Implementation of `filter` statements. Evaluation of SPARQL **`filter`** statements is pushed down to the underlying database D by translating filter expressions to appropriate SQL views. This allows to dynamically evaluate filter expressions on the DBMS side. For instance, given a rule $r \in D_Q$ of the form

```
h(X,Y) :- b(X,Y), &eval[f_Y](bool).
```

where the `&eval` atom encodes the **`filter`** statement (*f_Y* representing the filter expression), then r is translated to

```
h(X,Y) :- b'(X,Y).
```

where `b'` is a fresh predicate associated via the mapping file to a database view. Such a view defines the SQL code to be used for the computation of *f_Y*, like

```
CREATE VIEW B' AS ( SELECT X,Y FROM B WHERE F_Y )
```

where *F_Y* is an appropriate translation of the SPARQL **`filter`** expression *f_Y* at hand to an SQL Boolean condition,[10] while B is the DBMS counterpart table of the predicate b.

[10]A version of this translation can be found in [29].

5 Experiments

In order to illustrate that our approach is practically feasible, we present a quantitative performance comparison between our prototype system, GiaBATA, which implements the approach outlined before, and some state-of-the-art triple stores. The test were done on an Intel P4 3GHz machine with 1.5GB RAM under Linux 2.6.24. Let us briefly outline the main features and versions of the triple stores we used in our comparison.

AllegroGraph works as a database and application framework for building Semantic Web applications. The system assures persistent storage and RDFS++ reasoning, a semantic extension including the RDF and RDFS constructs and some OWL constructs (`owl:sameAs`, `owl:inverseOf`, `owl:TransitiveProperty`, `owl:hasValue`). We tested the free Java edition of AllegroGraph 3.2 with its native persistence mechanism.[11]

ARQ is a query engine implementing SPARQL under the Jena framework.[12] It can be deployed on several persistent storage layers, like filesystem or RDBMS, and it includes a rule-based inference engine. Being based on the Jena library, it provides inferencing models and enables (incomplete) OWL reasoning. Also, the system comes with support for custom rules. We used **ARQ** 2.6 with RDBMS backend connected to PostgreSQL 8.3.

GiaBATA [15] is our prototype system implementing the SPARQL extensions described above. GiaBATA is based on a combination of the DLV^{DB} [25] and dlvhex [24] systems, and caters for persistent storage of both data and ontology graphs. The former system is a variant of DLV [13] with built-in database support. The latter is a solver for HEX-programs [24], which features an extensible plugin system which we used for developing a rewriter-plugin able to translate SPARQL queries to HEX-programs. The tests were done using development versions of the above systems connected to PostgreSQL 8.3.

Sesame is an open source RDF database with support for querying and reasoning.[13] In addition to its in-memory database engine it can be coupled with relational databases or deployed on top of file systems. Sesame supports RDFS inference and other entailment regimes such as OWL-Horst [5] by coupling with external reasoners. Sesame provides an infrastructure for defining custom inference rules. Our tests have been done using Sesame 2.3 with persistence support given by the native store.

First of all, it is worth noting that all systems allow persistent storage on RDBMS. All systems, with the exception of ours, implement also direct filesystem storage. All cover RDFS (actually, disregarding axiomatic triples) and partial or non-standard OWL fragments. Although all the systems feature some form of persistence, both reasoning and query evaluation are usually performed in main memory. All the systems, except AllegroGraph and ours, adopt a persistent materialization approach for inferring data.

All systems – along with basic inference – support named graph querying, but, with the exception of GiaBATA, combining both features results in incomplete behavior as described in Section 2. Inference is properly handled as long as the query ranges over the whole dataset, whereas it fails in case of querying explicit default or named graphs.

[11] System available at `http://agraph.franz.com/allegrograph/`.
[12] Distributed at `https://jena.svn.sourceforge.net/svnroot/jena/ARQ/`.
[13] System available at `http://www.openrdf.org/`.

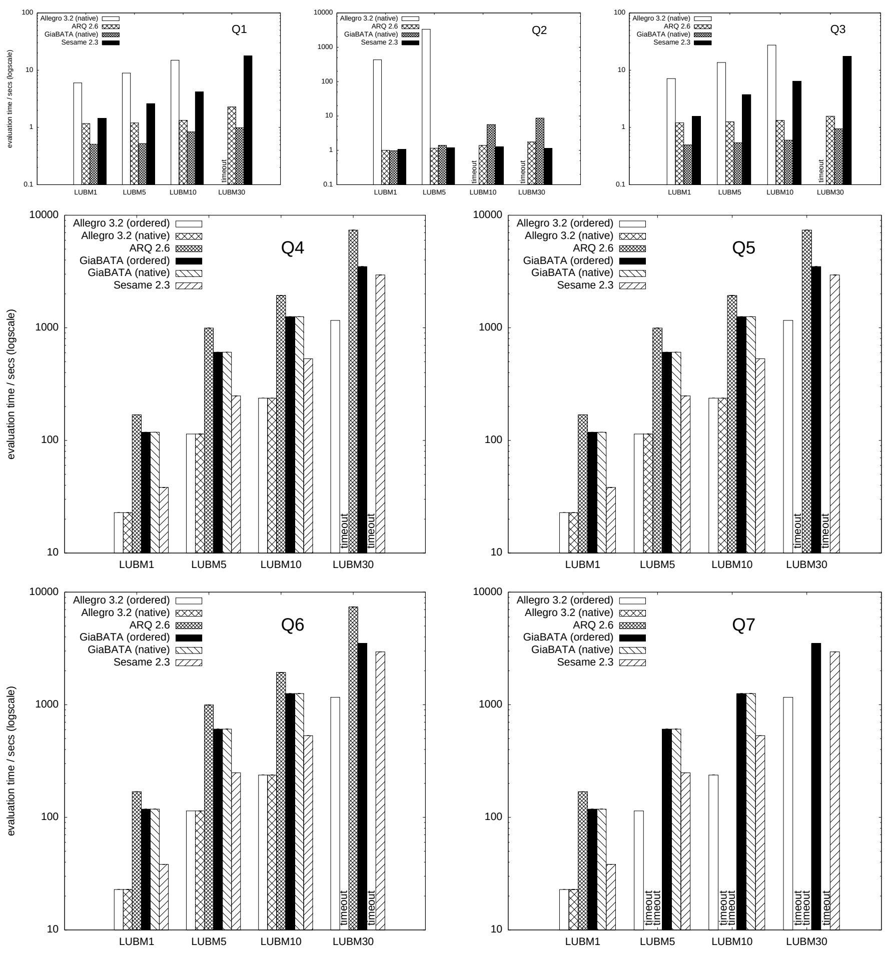


Figure 2: Evaluation

That makes querying of named graphs involving inference impossible with standard systems.

For performance comparison we rely on the LUBM benchmark suite [16]. Our tests involve the test datasets LUBMn for $n \in \{1, 5, 10, 30\}$ with LUBM30 having roughly four million triples (exact numbers are reported in [16]). In order to test the additional performance cost of our extensions, we opted for showing how the performance figures change when queries which require RDFS entailment rules (LUBM Q4-Q7) are considered, w.r.t. queries in which rules do not have an impact (LUBM Q1-Q3, see Appendix of [16] for the SPARQL encodings of $Q1$–$Q7$). Experiments are enough for comparing performance trends, so we didn't consider at this stage larger instances of LUBM. Note that evaluation times include the data loading times. Indeed, while former performance benchmarks do not take this aspect in account, from the semantic point of view, pre-materialization-at-loading computes inferences needed for complete query answering under the entailment of choice. On further reflection, dynamic querying of RDFS moves inference from this materialization to the query step, which would result in an apparent advantage for systems that rely on pre-materialization for RDFS data. Also, the setting of this paper assumes materialization cannot be performed *una tantum*, since inferred information depends on the entailment regime of choice, and on the dataset at hand, on a *per query* basis. We set a 120min query timeout limit to all test runs.

Our test runs include the following system setup: (i) "Allegro (native)" and "Allegro (ordered)" (ii) "ARQ"; (iii) "GiaBATA (native)" and "GiaBATA (ordered)"; and (iv) "Sesame". For (i) and (iii), which apply dynamic inference mechanisms, we use "(native)" and "(ordered)" to distinguish between executions of queries in LUBM's native ordering and in a optimized reordered version, respectively. The GiaBATA test runs both use Magic Sets optimization. To appreciate the cost of RDFS reasoning for queries $Q4$–$Q7$, the test runs for (i)–(iv) also include the loading time of the datasets, i.e., the time needed in order to perform RDFS data materialization or to simply store the raw RDF data.

The detailed outcome of the test results are summarized in Fig. 2. For the RDF test queries $Q1$–$Q3$, GiaBATA is able to compete for $Q1$ and $Q3$. The systems ARQ and Sesame turned out to be competitive for $Q2$ by having the best query response times, while Allegro (native) scored worst. For queries involving inference ($Q4$–$Q7$) Allegro shows better results. Interestingly, systems applying dynamic inference, namely Allegro and GiaBATA, query pattern reordering plays a crucial role in preserving performance and in assuring scalability; without reordering the queries simply timeout. In particular, Allegro is well-suited for queries ranging over several properties of a single class, whereas if the number of classes and properties increases ($Q7$), GiaBATA exhibits better scalability. Finally, a further distinction between systems relying on DBMS support and systems using native structures is disregarded, and since figures (in logarithmic scale) depict overall loading and querying time, this penalizes in specific cases those systems that use a DBMS.

6 Future Work and Conclusion

We presented a framework for dynamic querying of RDFS data, which extends SPARQL by two language constructs: **`using ontology`** and **`using ruleset`**. The former is geared towards dynamically creating the dataset, whereas the latter adapts the entailment regime of the query. We have shown that our extension conservatively extends the standard SPARQL language and that by selecting appropriate rules in **`using ruleset`**, we may choose varying rule-based entailment regimes at query-time. We illustrated how such extended SPARQL queries can be translated to Datalog and SQL, thus providing entry points for implementation and well-known optimization techniques. Our initial experiments have shown that although dynamic querying does more computation at query-time, it is still competitive for use cases that need on-the-fly construction of datasets and entailment regimes. Especially here, query optimization techniques play a crucial role, and our results suggest to focus further research in this direction. Furthermore, we aim at conducting a proper computational analysis as it has been done for Hypothetical datalog [30], in which truth of atoms is conditioned by hypothetical additions to the dataset at hand. Likewise, our framework allows to add ontological knowledge and rules to datasets before querying: note however that, in the spirit of [31], our framework allows for hypotheses (also called “premises”) on a per query basis rather than a per atom basis.

References

[1] Klyne, G., Carroll, J.J. (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Rec. (February 2004)

[2] Brickley, D., Miller, L.: FOAF Vocabulary Specification 0.91 (2007) `http://xmlns.com/foaf/spec/`.

[3] Bojārs, U., Breslin, J.G., Berrueta, D., Brickley, D., Decker, S., Fernández, S., Görn, C., Harth, A., Heath, T., Idehen, K., Kjernsmo, K., Miles, A., Passant, A., Polleres, A., Polo, L., Sintek, M.: SIOC Core Ontology Specification (June 2007) W3C member submission.

[4] Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: WWW2007, ACM (2007)

[5] ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. Web Semant. **3**(2–3) (2005) 79–115

[6] Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. In: ESWC’07. Springer (2007) 53–67

[7] Hogan, A., Harth, A., Polleres, A.: Scalable authoritative owl reasoning for the web. Int. J. Semant. Web Inf. Syst. **5**(2) (2009)

[8] Polleres, A., Scharffe, F., Schindlauer, R.: SPARQL++ for mapping between RDF vocabularies. In: ODBASE’07. Springer (2007) 878–896

[9] Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In: ISWC’07. Springer (2007) 211–224

[10] Prud’hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Rec. (January 2008)

[11] Polleres, A.: From SPARQL to rules (and back). In: WWW2007. ACM (2007) 787–796

[12] Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: ISWC’08. Springer (2008) 114–129

[13] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3) (2006) 499–562

[14] Euzenat, J., Polleres, A., Scharffe, F.: Processing ontology alignments with SPARQL. In: OnAV’08 Workshop, CISIS’08, IEEE Computer Society (2008) 913–917

[15] Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: A Rule System for Querying Persistent RDFS Data. In : ESWC'09. Springer (2009) 857–862

[16] Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. J. Web Semant. **3**(2–3) (2005) 158–182

[17] Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language Profiles W3C Cand. Rec. (June 2009)

[18] Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. In: ISWC'06. Springer (2006) 30–43

[19] Polleres, A., Schindlauer, R.: dlvhex-sparql: A SPARQL-compliant query engine based on dlvhex. In: ALPSWS2007. CEUR-WS (2007) 3–12

[20] Hayes, P.: RDF semantics. W3C Rec. (February 2004).

[21] Ianni, G., Martello, A., Panetta, C., Terracina, G.: Efficiently querying RDF(S) ontologies with answer set programming. J. Logic Comput. **19**(4) (2009) 671–695

[22] de Bruijn, J.: Semantic Web Language Layering with Ontologies, Rules, and Meta-Modeling. PhD thesis, University of Innsbruck (2008)

[23] Boley, H., Kifer, M.: RIF Basic Logic Dialect. W3C Working Draft (July 2009)

[24] Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic web reasoning. In: ESWC'06. Springer (2006) 273–287

[25] Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory Pract. Log. Program. **8**(2) (2008) 129–165

[26] Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: ISWC'05. Springer (2005) 685–701

[27] Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB. ACM (2007) 411–422

[28] Beeri, C., Ramakrishnan, R.: On the power of magic. J. Log. Program. **10**(3-4) (1991) 255–299

[29] Lu, J., Cao, F., Ma, L., Yu, Y., Pan, Y.: An Effective SPARQL Support over Relational Databases. In: SWDB-ODBIS. (2007) 57–76

[30] Bonner, A.J.: Hypothetical datalog: complexity and expressibility. Theor. Comp. Sci. **76**(1) (1990) 3–51

[31] Gutiérrez, C., Hurtado, C.A., Mendelzon, A.O.: Foundations of semantic web databases. In: PODS 2004, ACM (2004) 95–106

Scalable Authoritative OWL Reasoning for the Web
Aidan Hogan, Andreas Harth, and Axel Polleres
Published in International Journal on Semantic Web and Information Systems (IJSWIS), Volume 5, Number 2, pp. 49–90, May 2009, IGI Global, ISSN 1552-6283

Scalable Authoritative OWL Reasoning for the Web [*]

Aidan Hogan *Andreas Harth* *Axel Polleres*

Digital Enterprise Research Institute
National University of Ireland, Galway

Abstract

In this paper we discuss the challenges of performing reasoning on large scale RDF datasets from the Web. Using ter-Horst's pD* fragment of OWL as a base, we compose a rule-based framework for application to web data: we argue our decisions using observations of undesirable examples taken directly from the Web. We further temper our OWL fragment through consideration of "authoritative sources" which counter-acts an observed behaviour which we term "ontology hijacking": new ontologies published on the Web re-defining the semantics of existing entities resident in other ontologies. We then present our system for performing rule-based forward-chaining reasoning which we call SAOR: *Scalable Authoritative OWL Reasoner*. Based upon observed characteristics of web data and reasoning in general, we design our system to scale: our system is based upon a separation of terminological data from assertional data and comprises of a lightweight in-memory index, on-disk sorts and file-scans. We evaluate our methods on a dataset in the order of a hundred million statements collected from real-world web sources and present scale-up experiments on a dataset in the order of a billion statements collected from the Web.

1 Introduction

Information attainable through the Web is unique in terms of scale and diversity. The Semantic Web movement aims to bring order to this information by providing a stack of technologies, the core of which is the Resource Description Framework (RDF) for publishing data in a machine-readable format: there now exists millions of RDF data-sources on the Web contributing billions of statements. The Semantic Web technology stack includes means to supplement instance data being published in RDF with

[*]A preliminary version of this article has been accepted at ASWC 2008 [24]. Compared to that version, we have added significant material. The added contributions in this version include (i) a better formalisation of authoritative reasoning, (ii) improvements in the algorithms, and (iii) respectively updated experimental results with additional metrics on a larger dataset. We thank the anonymous reviewers of this and related papers for their valuable feedback. This work has been supported by Science Foundation Ireland project Lion (SFI/02/CE1/I131), European FP6 project inContext (IST-034718), COST Action "Agreement Technologies" (IC0801) and an IRCSET Postgraduate Research Scholarship.

ontologies described in RDF Schema (RDFS) [4] and the Web Ontology Language (OWL) [2, 41], allowing people to formally specify a domain of discourse, and providing machines a more sapient understanding of the data. In particular, the enhancement of assertional data (i.e., instance data) with terminological data (i.e., structural data) published in ontologies allows for deductive reasoning: i.e., inferring implicit knowledge.

In particular, our work on reasoning is motivated by the requirements of the Semantic Web Search Engine (SWSE) project: `http://swse.deri.org/`, within which we strive to offer search, querying and browsing over data taken from the Semantic Web. Reasoning over aggregated web data is useful, for example: to infer new assertions using terminological knowledge from ontologies and therefore provide a more complete dataset; to unite fractured knowledge (as is common on the Web in the absence of restrictive formal agreement on identifiers) about individuals collected from disparate sources; and to execute mappings between domain descriptions and thereby provide translations from one conceptual model to another. The ultimate goal here is to provide a "global knowledge-base", indexed by machines, providing querying over both the explicit knowledge published on the Web and the implicit knowledge inferable by machine. However, as we will show, complete inferencing on the Web is an infeasible goal, due firstly to the complexity of such a task and secondly to noisy web data; we aim instead to strike a comprise between the above goals for reasoning and what is indeed feasible for the Web.

Current systems have had limited success in exploiting ontology descriptions for reasoning over RDF web data. While there exists a large body of work in the area of reasoning algorithms and systems that work and scale well in confined environments, the distributed and loosely coordinated creation of a world-wide knowledge-base creates new challenges for reasoning:

- the system has to perform on web scale, with implications on the completeness of the reasoning procedure, algorithms and optimisations;
- the method has to perform on collaboratively created knowledge-bases, which has implications on trust and the privileges of data publishers.

With respect to the first requirement, many systems claim to inherit their scalability from the underlying storage – usually some relational database system – with many papers having been dedicated to optimisations on database schemata and access (c.f. [35, 44, 48, 25]). With regards the second requirement, there have been numerous papers dedicated to the inter-operability of a small number of usually trustworthy ontologies (c.f. [13, 31, 27]). We leave further discussion of related work to Section 6, except to state that the combination of web scale and web tolerant reasoning has received little attention in the literature and that our approach is novel.

Our system, which we call "Scalable Authoritative OWL Reasoner" (SAOR), is designed to accept as input a web knowledge-base in the form of a body of statements as produced by a web crawl and to output a knowledge-base enhanced by forward-chaining reasoning over a given fragment of OWL. In particular, we choose forward-chaining to avoid the runtime complexity of query-rewriting associated with backward-chaining approaches: in the web search scenario, the requirement for low query response times and resource usage preclude the applicability of query-rewriting for many reasoning tasks.

SAOR adopts a standard rule-based approach to reasoning whereby each rule consists of (i) an 'antecedent': a clause which identifies a graph pattern that, when matched by the data, allows for the rule to be executed and (ii) a 'consequent': the statement(s) that can be inferred given data that match the antecedent. Within SAOR, we view reasoning as a once-off rule-processing task over a given set of statements. Since the rules are all known *a-priori*, and all require simultaneous execution, we can design a task-specific system that offers much greater optimisations over more general rule engines. Firstly, we categorise the known rules according to the composition of their antecedents (e.g., with respect to arity, proportion of terminological and assertional patterns, etc.) and optimise each group according to the observed characteristics. Secondly, we do not use an underlying database or native RDF store and opt for implementation using fundamental data-structures and primitive operations; our system is built from scratch specifically (and only) for the purpose of performing pre-runtime forward-chaining reasoning which gives us greater freedom in implementing appropriate task-specific optimisations.

This paper is an extended version of [24], in which we presented an initial *modus-operandi* of SAOR; we provided some evaluation of a set of rules which exhibited linear scale and concluded that using dynamic index structures, in SAOR, for more complex rulesets, was not a viable solution for a large-scale reasoner. In this paper, we provide extended discussion of our fragment of OWL reasoning and additional motivation for our deliberate incompleteness in terms of computational complexity and impediments posed by web data considerations. We also describe an implementation of SAOR which abandons dynamic index structures in favour of batch processing techniques known to scale: namely sorts and file-scans. We present new evaluation of the adapted system over a dataset of 147m triples collected from 665k web sources and also provide scale-up evaluation of our most optimised ruleset on a dataset of 1.1b statements collected from 6.5m web sources.

Specifically, we make the following contributions in this paper:

- We discuss and apply a selected rule-based subset of OWL reasoning, i) to be computationally efficient, ii) to avoid an explosion of inferred statements, iii) to be tolerant to noisy web data and iv) to protect existing specifications from undesirable contributions made in independent locations. That is, our system implements a positive fragment of OWL Full which has roots in ter Horst's pD* [43] entailment rules and our system includes analysis of the authority of sources to counter-act the problem of *ontology hijacking* in web data (Section 3).

- We describe a scalable, optimised method for performing rule-based forward-chaining reasoning for our fragment of OWL. In particular, we refine our algorithm to capitalise on the similarities present in different rule antecedent patterns and the low volume of terminological data relative to assertional data. We implement the system using on-disk batch processing operations known to scale: sorts and scans (Section 4).

- We show experimentally that a forward-chaining materialisation approach is feasible on web data, showing that, by careful materialisation through our tailored OWL ruleset, we can avoid an explosion of inferred statements. We present evaluation with respect to computation of our most expressive ruleset on a dataset of 147m statements collected from 665k sources and present scale-up measurements by applying our most optimised ruleset on a dataset of 1.1b statements collected from 6.5m sources. We also reveal that the most computationally efficient segment of our reasoning is the most productive with regards inferred output statements (Section 5).

We discuss related work in Section 6 and conclude with Section 7.

2 Preliminaries

Before we continue, we briefly introduce some concepts prevalent throughout the paper. We use notation and nomenclature as is popular in the literature, particularly from [22].

RDF Term Given a set of URI references $\mathcal{U}$, a set of blank nodes $\mathcal{B}$, and a set of literals $\mathcal{L}$, the set of *RDF terms* is denoted by $\mathcal{RDFT}erm = \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. The set of blank nodes $\mathcal{B}$ is a set of existensially quantified variables. The set of literals is given as $\mathcal{L} = \mathcal{L}_p \cup \mathcal{L}_t$, where $\mathcal{L}_p$ is the set of *plain literals* and $\mathcal{L}_t$ is the set of *typed literals*. A typed literal is the pair $l = (s,t)$, where s is the lexical form of the literal and $t \in \mathcal{U}$ is a datatype URI. The sets $\mathcal{U}$, $\mathcal{B}$, $\mathcal{L}_p$ and $\mathcal{L}_t$ are pairwise disjoint.

RDF Triple A triple $t = (s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called an *RDF triple*. In a triple (s, p, o), s is called subject, p predicate, and o object.

RDF Triple in Context/RDF Quadruple A pair (t, c) with a triple $t = (s, p, o)$ and $c \in \mathcal{U}$ is called a *triple in context* c [16, 20]. We may also refer to (s, p, o, c) as the *RDF quadruple* or quad q with context c.

We use the term 'RDF statement' to refer generically to triple or quadruple where differentiation is not pertinent.

RDF Graph/Web Graph An *RDF graph* $\mathcal{G}$ is a set of RDF triples; that is, a subset of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$.

We refer to a *web graph* $\mathcal{W}$ as a graph derived from a given web location (i.e., a given document). We call the pair $(\mathcal{W}, c)$ a web graph $\mathcal{W}$ in context c, where c is the web location from which $\mathcal{W}$ is retrieved. Informally, $(\mathcal{W}, c)$ is represented as the set of quadruples (t_w, c) for all $t_w \in \mathcal{W}$.

Generalised Triple A triple $t = (s, p, o) \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called a *generalised triple*.

The notions of generalised quadruple, generalised statement and generalised graph follow naturally. Our definition of "generalised" is even more liberal than that described in [43] wherein blank nodes are allowed in the predicate position: we also allow literals in the subject and predicate position. Please note that we may refer generically to a "triple", "quadruple", "graph" etc. where a distinction between the "generalised" and "RDF" versions is not pertinent.

Merge The *merge* M($\mathcal{S}$) of a set of graphs $\mathcal{S}$ is the union of the set of all graphs $\mathcal{G}'$ for $\mathcal{G} \in \mathcal{S}$ and $\mathcal{G}'$ derived from $\mathcal{G}$ such that $\mathcal{G}'$ contains a unique set of blank nodes for $\mathcal{S}$.

Web Knowledge-base Given a set $\mathcal{S}_\mathcal{W}$ of RDF web graphs, our view of a *web knowledge-base* $\mathbb{KB}$ is taken as a set of pairs ($\mathcal{W}'$, *c*) for each $\mathcal{W} \in \mathcal{S}_\mathcal{W}$, where $\mathcal{W}'$ contains a unique set of blank nodes for $\mathcal{S}_\mathcal{W}$ and *c* denotes the URL location of $\mathcal{W}$.

Informally, $\mathbb{KB}$ is a set of quadruples retrieved from the Web wherein the set of blank nodes are unique for a given document and triples are enhanced by means of context which tracks the web location from which each triple is retrieved. We use the abbreviated notation $\mathcal{W} \in \mathbb{KB}$ or $\mathcal{W}' \in \mathbb{KB}$ where we mean $\mathcal{W} \in \mathcal{S}_\mathcal{W}$ for $\mathcal{S}_\mathcal{W}$ from which $\mathbb{KB}$ is derived or $(\mathcal{W}', c) \in \mathbb{KB}$ for some *c*.

Class We refer to a *class* as an RDF term which appears in either

- *o* of a triple *t* where *p* is `rdf:type`; or
- *s* of a triple *t* where *p* is `rdf:type` and *o* is `rdfs:Class` or `:Class`[1].

Property We refer to a *property* as an RDF term which appears in either

- *p* of a triple *t*; or
- *s* of a triple *t* where *p* is `rdf:type` and *o* is `rdf:Property`.

Membership Assertion We refer to a triple *t* as a *membership assertion* of the property mentioned in predicate position *p*. We refer to a triple *t* with predicate `rdf:type` as a membership assertion of the class mentioned in the object *o*. For a class or property *v*, we denote a membership assertion as *m(v)*.

Meta-class A *meta-class* is a class of classes or properties; i.e., the members of a meta-class are either classes or properties. The set of RDF(S) and OWL meta-classes is as follows: {`rdf:Property`, `rdfs:Class`, `rdfs:ContainerMembershipProperty`, `:AnnotationProperty`, `:Class`, `:DatatypeProperty`, `:DeprecatedClass`, `:DeprecatedProperty`, `:FunctionalProperty`, `:InverseFunctionalProperty`, `:ObjectProperty`, `:OntologyProperty`, `:Restriction`, `:SymmetricProperty`, `:TransitiveProperty` }.

Meta-property A *meta-property* is one which has a meta-class as it's domain. Meta-properties are used to describe classes and properties. The set of RDFS and OWL meta-properties is as follows: {`rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `:allValuesFrom`, `:cardinality`, `:complementOf`, `:disjointWith`, `:equivalentClass`, `:equivalentProperty`, `:hasValue`, `:intersectionOf`, `:inverseOf`, `:maxCardinality`, `:minCardinality`, `:oneOf`, `:onProperty`, `:someValuesFrom`, `:unionOf`}.

[1]Throughout this paper, we assume that `http://www.w3.org/2002/07/owl#` is the default namespace with prefix "`:`", i.e. we write e.g. just "`:Class`", "`:disjointWith`", etc. instead of using the commonly used `owl:` prefix. Other prefixes such as `rdf:`, `rdfs:`, `foaf:` are used as in other common documents. Moreover, we often use the common abbreviation '`a`' as a convenient shortcut for `rdf:type`.

Terminological Triple We define a *terminological triple* as one of the following:

1. a membership assertion of a meta-class;
2. a membership assertion of a meta-property;
3. a triple in a non-branching, non-cyclic path $t_0^r, ..., t_n^r$ where $t_0^r = (s_0, p_0, o_0)$ for $p_0 \in \{$`:intersectionOf`, `:oneOf`, `:unionOf` $\}$; $t_k^r = (o_{k-1},$ `rdf:rest`, $o_k)$ for $1 \leq k \leq n$, $o_{k-1} \in \mathcal{B}$ and $o_n =$`rdf:nil`; or a triple $t_k^f = (o_k,$ `rdf:first`, $e_k)$ with o_k for $0 \leq k < n$ as before.

We refer to triples $t_1^r, ..., t_n^r$ and all triples t_k^f as *terminological collection triples*, whereby RDF collections are used in a union, intersection or enumeration class description.

Triple Pattern, Basic Graph Pattern A *triple pattern* is defined as a generalised triple where, in all positions, variables from the infinite set $\mathcal{V}$ are allowed; i.e.: *tp* = $(s_v, p_v, o_v) \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$. A set (to be read as conjunction) of triple patterns $\mathcal{GP}$ is also called a *basic graph pattern*.

We use – following SPARQL notation [38] – alphanumeric strings preceded by '`?`' to denote variables in this paper: e.g., `?X`. Following common notation, such as is used in SPARQL [38] and Turtle[2], we delimit triples in the same basic graph pattern by '.' and we may group triple patterns with the same subject or same subject-predicate using ';' and ',' respectively. Finally, we denote by $\mathcal{V}(tp)$ (or $\mathcal{V}(\mathcal{GP})$, resp.) the set of variables appearing in tp (or in $\mathcal{GP}$, resp.).

Instance A triple *t* = (*s*, *p*, *o*) (or, resp., a set of triples, i.e., a graph $\mathcal{G}$) is an *instance* of a triple pattern *tp* = (s_v, p_v, o_v) (or, resp., of a basic graph pattern $\mathcal{GP}$) if there exists a mapping $\mu : \mathcal{V} \cup \mathcal{RDFT}erm \rightarrow \mathcal{RDFT}erm$ which maps every element of $\mathcal{RDFT}erm$ to itself, such that $t = \mu(tp) = (\mu(s_v), \mu(p_v), \mu(o_v))$ (or, resp., and slightly simplifying notation, $G = \mu(\mathcal{GP})$).

Terminological/Assertional Pattern We refer to a *terminological -triple/-graph pattern* as one whose instance can only be a terminological triple or, resp., a set thereof. We denote a *terminological collection pattern* by `?x` p `(?e`$_1$`, ..., ?e`$_n$`) .` where $p \in \{$`:intersectionOf`, `:oneOf`, `:unionOf` $\}$ and `?e`$_k$ is mapped by the object of a terminological collection triple (o$_k$, `rdf:first`, e$_k$) for $o_k \in \{o_0, ..., o_{n-1}\}$ as before. An *assertional pattern* is any pattern which is not terminological.

Inference Rule We define an *inference rule* r as the pair $(\mathcal{A}nte, \mathcal{C}on)$, where the *antecedent* $\mathcal{A}nte$ and the *consequent* $\mathcal{C}on$ are basic graph patterns such that $\mathcal{V}(\mathcal{C}on)$ and $\mathcal{V}(\mathcal{A}nte)$ are non-empty, $\mathcal{V}(\mathcal{C}on) \subseteq \mathcal{V}(\mathcal{A}nte)$ and $\mathcal{C}on$ does not contain blank nodes[3]. In this paper, we will typically write inference rules as:

$$\mathcal{A}nte \Rightarrow \mathcal{C}on \tag{1}$$

[2] `http://www.dajobe.org/2004/01/turtle/`

[3] Unlike some other rule systems for RDF, the most prominent of which being CONSTRUCT statements in SPARQL, we forbid blank nodes; i.e., we forbid existential variables in rule consequents which would require the "invention" of blank nodes.

Rule Application and Closure We define a *rule application* in terms of the immediate consequences of a rule r or a set of rules $\mathcal{R}$ on a graph $\mathcal{G}$ (here slightly abusing the notion of the immediate consequence operator in Logic Programming: cf. for example [30]). That is, if r is a rule of the form (1), and $\mathcal{G}$ is a set of RDF triples, then:

$$T_r(\mathcal{G}) = \{\mu(\mathcal{C}on) \mid \exists\mu \text{ such that } \mu(\mathcal{A}nte) \subseteq G\}$$

and accordingly $T_{\mathcal{R}}(\mathcal{G}) = \bigcup_{r\in\mathcal{R}} T_r(\mathcal{G})$. Also, let $\mathcal{G}_{i+1} = \mathcal{G}_i \cup T_{\mathcal{R}}(\mathcal{G}_i)$ and $\mathcal{G}_0 = \mathcal{G}$; we now define the *exhaustive application* of the $T_{\mathcal{R}}$ operator on a graph $\mathcal{G}$ as being upto the least fixpoint (the smallest value for n) such that $\mathcal{G}_n = T_{\mathcal{R}}(\mathcal{G}_n)$. We call $\mathcal{G}_n$ the *closure* of $\mathcal{G}$ with respect to ruleset $\mathcal{R}$, denoted as $Cl_{\mathcal{R}}(\mathcal{G})$. Note that we may also use the intuitive notation $Cl_{\mathcal{R}}(\mathbb{KB})$, $T_{\mathcal{R}}(\mathbb{KB})$ as shorthand for the more cumbersome $Cl_{\mathcal{R}}(\bigcup_{\mathcal{W}'\in\mathbb{KB}} \mathcal{W}')$, $T_{\mathcal{R}}(\bigcup_{\mathcal{W}'\in\mathbb{KB}} \mathcal{W}')$.

Ground Triple/Graph A *ground triple* or *ground graph* is one without existential variables.

Herbrand Interpretation Briefly, a *Herbrand interpretation* of a graph $\mathcal{G}$ treats URI references, blank nodes, typed literals and plain literals analogously as denoting their own syntactic form. As such, a Herbrand interpretation represents a ground view of an RDF graph where blank nodes are treated as *Skolem* names instead of existential variables; i.e., blank nodes are seen to represent the entities that they assert the existence of, analogously to a URI reference. Henceforth, we view blank nodes as their Skolem equivalents (this also applies to blank nodes as mentioned in the above notation) and only treat the ground case of RDF graphs.

Let us elaborate in brief why this treatment of blank nodes as Skolem constants is sufficient for our purposes. In our scenario, we perform forward-chaining materialisation for query-answering and not "real" entailment checks between RDF graphs. This enables us to treat all blank nodes as Skolem names [22]. It is well known that simple entailment checking of two RDF graphs [22] – i.e., checking whether an RDF graph $\mathcal{G}_1$ entails $\mathcal{G}_2$ – can be done using the ground "skolemised" version of $\mathcal{G}_1$. That is $\mathcal{G}_1 \models \mathcal{G}_2$ iff $sk(\mathcal{G}_1) \models \mathcal{G}_2$. Likewise, given a set of inference rules $\mathcal{R}$, where we denote entailment with respect to $\mathcal{R}$ as $\models_{\mathcal{R}}$, it is again well known that such entailment can be reduced to simple entailment with prior computation of the inference closure with respect to $\mathcal{R}$. That is, $\mathcal{G}_1 \models_{\mathcal{R}} \mathcal{G}_2$ iff $Cl_{\mathcal{R}}(sk(\mathcal{G}_1)) \models \mathcal{G}_2$, cf. [22, 18]. In this paper we focus on the actual computation of $Cl_{\mathcal{R}}(sk(\mathcal{G}_1))$ for a tailored ruleset $\mathcal{R}$ in between RDFS and OWL Full.

3 Pragmatic Inferencing for the Web

In this section we discuss the inference rules which we use to approximate OWL semantics and are designed for forward-chaining reasoning over web data. We justify our selection of inferences to support in terms of observed characteristics and examples taken from the Web. We optimise by restricting our fragment of reasoning according to three imperatives: *computational feasibility (CF)* for scalability, *reduced output* statements *(RO)* to ease the burden on consumer applications and, finally, *web tolerance (WT)* for avoiding undesirable inferences given noisy data and protecting publishers from unwanted, independent third-party contributions. In particular, we adhere to the following high-level restrictions:

1. we are incomplete *(CF, RO, WT)* - Section 3.1;
2. we deliberately ignore the explosive behaviour of classical inconsistency *(CF, RO, WT)* - Section 3.1;
3. we follow a rule-based, finite, forward-chaining approach to OWL inference *(CF)* - Section 3.2;
4. we do not invent new blank nodes *(CF, RO, WT)* - Section 3.2;
5. we avoid inference of *extended-axiomatic* triples *(RO)* - Section 3.2;
6. we focus on inference of non-terminological statements *(CF)* - Section 3.2;
7. we do not consider `:sameAs` statements as applying to terminological data *(CF, WT)* - Section 3.2;
8. we separate and store terminological data in-memory *(CF)* - Section 3.3;
9. we support limited reasoning for *non-standard use* of the RDF(S) and OWL vocabularies *(CF, RO, WT)* - Section 3.3;
10. we ignore *non-authoritative* (third-party) terminological statements from our reasoning procedure to counter an explosion of inferred statements caused by *hijacking* ontology terms *(RO, WT)* - Section 3.4.

3.1 Infeasibility of Complete Web Reasoning

Reasoning over RDF data is enabled by the description of RDF terms using the RDFS and OWL standards; these standards have defined entailments determined by their semantics. The semantics of these standards differs in that RDFS entailment is defined in terms of "if" conditions (intensional semantics), and has a defined set of complete standard entailment rules [22]. OWL semantics uses "iff" conditions (extensional semantics) without a complete set of standard entailment rules. RDFS entailment has been shown to be decidable and in P for the ground case [43], whilst OWL Full entailment is known to be undecidable [26]. Thus, the OWL standard includes two restricted fragments of OWL whose entailment is known to be decidable from work in description logics: (i) OWL DL whose worst-case entailment is in NEXPTIME (ii) OWL Lite whose worst-case entailment is in EXPTIME [26].

Although entailment for both fragments is known to be decidable, and even aside from their complexity, most OWL ontologies crawlable on the Web are in any case OWL Full: idealised assumptions made in OWL DL are violated by even very commonly used ontologies. For example, the popular Friend Of A Friend (FOAF) vocabulary [5] deliberately falls into OWL Full since, in the FOAF RDF vocabulary[4], `foaf:name` is defined as a sub-property of the core RDFS property `rdfs:label` and `foaf:mbox_sha1sum` is defined as both an `:InverseFunctionalProperty` and a `:DatatypeProperty`: both are disallowed by OWL DL (and, of course,

[4]`http://xmlns.com/foaf/spec/index.rdf`

OWL Lite). In [3], the authors identified and categorised OWL DL restrictions violated by a sample group of 201 OWL ontologies (all of which were found to be in OWL Full); these include incorrect or missing typing of classes and properties, complex object-properties (e.g., functional properties) declared to be transitive, inverse-functional datatype properties, etc. In [46], a more extensive survey with nearly 1,300 ontologies was conducted: 924 were identified as being in OWL Full.

Taking into account that most web ontologies are in OWL Full, and also the undecidability/computational infeasiblity of OWL Full, one could conclude that complete reasoning on the Web is impractical. However, again for most web documents only categorisable as OWL Full, infringements are mainly syntactic and are rather innocuous with no real effect on decidability ([46] showed that the majority of web documents surveyed were in the base expressivity for Description Logics after patching infringements).

The main justification for the infeasibility of complete reasoning on the Web is inconsistency.

Consistency cannot be expected on the Web; for instance, a past web crawl of ours revealed the following:

```
w3:timbl a foaf:Person; foaf:homepage <http://w3.org/> .
w3:w3c a foaf:Organization; foaf:homepage <http://w3.org/>
.
foaf:homepage a :InverseFunctionalProperty .
foaf:Organization :disjointWith foaf:Person .
```

These triples together infer that Tim Berners-Lee is the same as the W3C and thus cause an inconsistency.[5] Aside from such examples which arise from misunderstanding of the FOAF vocabulary, there might be cases where different parties deliberately make contradictive statements; resolution of such contradictions could involve "choosing sides". In any case, the explosive nature of contradiction in classical logics suggests that it is not desirable within our web reasoning scenario.

3.2 Rule-based Web Reasoning

As previously alluded to, there does not exist a standard entailment for OWL suitable to our web reasoning scenario. However, incomplete (wrt. OWL Full) rule-based inference (i.e., reasoning as performed by logic progamming or deductive database engines) may be considered to have greater potential for scale, following the arguments made in [12] and may be considered to be more robust with respect to preventing explosive inferencing through inconsistencies. Several rule expressible non-standard OWL fragments; namely OWL-DLP [15], OWL$^-$ [10] (which is a slight extension of OWL-DLP), OWLPrime [47], pD* [42, 43], and Intensional OWL [9, Section 9.3]; have been defined in the literature and enable incomplete but sound RDFS and OWL Full inferences.

In [42, 43], pD* was introduced as a combination of RDFS entailment, datatype reasoning and a distilled version of OWL with rule-expressible intensional semantics: pD* entailment maintains the computational complexity of RDFS entailment, which is in NP in general and P for the ground case. Such improvement in complexity has

[5]Tim (now the same entity as the W3C) is asserted to be a member of the two disjoint classes: `foaf:Person` and `foaf:Organization`.

obvious advantages in our web reasoning scenario; thus SAOR's approach to reasoning is inspired by the pD* fragment to cover large parts of OWL by positive inference rules which can be implemented in a forward-chaining engine.

Table 1 summarises the pD* ruleset. The rules are divided into D*-entailment rules and P-entailment rules. D*-entailment is essentially RDFS entailment [22] combined with some datatype reasoning. P-entailment is introduced in [43] as a set of rules which applies to a property-related subset of OWL.

Given pD*, we make some amendments so as to align the ruleset with our requirements. Table 2 provides a full listing of our own modified ruleset, which we compare against pD* in this section. Note that this table highlights characteristics of the rules which we will discuss in Section 3.3 and Section 3.4; for the moment we point out that **rule′** is used to indicate an amendment to the respective pD* rule. Please also note that we use the notation **rulex*** to refer to all rules with the prefix **rulex**.

pD*	**rule**	**where**
	***D**-entailment rules**	
lg	?x ?P ?l . ⇒ ?v ?P _:*bl* .	?l$\in \mathcal{L}$[a]
gl	?x ?P _:*bl* . ⇒ ?x ?P ?l .	?l$\in \mathcal{L}$
rdf1	?x ?P ?y . ⇒ ?P a rdf:Property .	
rdf2-D	?x ?P ?l . ⇒ _:*bl* ?type ?t .	?l$= (s,t) \in \mathcal{L}_t$
rdfs1	?x ?P ?l . ⇒ _:*bl* a Literal .	?l$\in \mathcal{L}_p$
rdfs2	?P rdfs:domain ?C . ?x ?P ?y . ⇒ ?x a ?C .	
rdfs3	?P rdfs:range ?C . ?x ?P ?y . ⇒ ?y a ?C .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfs4a	?x ?P ?y . ⇒ ?x a rdfs:Resource .	
rdfs4b	?x ?P ?y . ⇒ ?y a rdfs:Resource .	?y$\in \mathcal{U} \cup \mathcal{B}$
rdfs5	?P rdfs:subProperty ?Q . ?Q rdfs:subProperty ?R . ⇒ ?P rdfs:subProperty ?R .	
rdfs6	?P a rdf:Property . ⇒ ?P rdfs:subProperty ?P .	
rdfs7	?P rdfs:subProperty ?Q . ?x ?P ?y . ⇒ ?x ?Q ?y .	?Q$\in \mathcal{U} \cup \mathcal{B}$
rdfs8	?C a rdfs:Class . ⇒ ?C rdfs:subClassOf rdfs:Resource .	
rdfs9	?C rdfs:subClassOf ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfs10	?C a rdfs:Class . ⇒ ?C rdfs:subClassOf ?C .	
rdfs11	?C rdfs:subClassOf ?D . ?D rdfs:subClassOf ?E . ⇒ ?C rdfs:subClassOf ?E .	
rdfs12	?P a rdfs:ContainerMembershipProperty . ⇒ ?P rdfs:subPropertyOf rdfs:member .	
rdfs13	?D a rdfs:Datatype . ⇒ ?D rdfs:subClassOf rdfs:Literal .	
	***P*-entailment rules**	
rdfp1	?P a :FunctionalProperty . ?x ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp2	?P a :InverseFunctionalProperty . ?x ?P ?z . ?y ?P ?z . ⇒ ?x :sameAs ?y .	
rdfp3	?P a :SymmetricProperty . ?x ?P ?y . ⇒ ?y ?P ?x .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp4	?P a :TransitiveProperty . ?x ?P ?y . ?y ?P ?z . ⇒ ?x ?P ?z .	
rdfp5a	?x ?P ?y . ⇒ ?x :sameAs ?x .	
rdfp5b	?x ?P ?y . ⇒ ?y :sameAs ?y .	?y$\in \mathcal{U} \cup \mathcal{B}$
rdfp6	?x :sameAs ?y . ⇒ ?y :sameAs ?x .	?y$\in \mathcal{U} \cup \mathcal{B}$
rdfp7	?x :sameAs ?y . ?y :sameAs ?z . ⇒ ?x :sameAs ?z .	
rdfp8a	?P :inverseOf ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	?y,?Q$\in \mathcal{U} \cup \mathcal{B}$
rdfp8b	?P :inverseOf ?Q . ?x ?Q ?y . ⇒ ?y ?P ?x .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp9	?C a :Class ; :sameAs ?D . ⇒ ?C rdfs:subClassOf ?D .	
rdfp10	?P a :Property ; :sameAs ?Q . ⇒ ?P rdfs:subPropertyOf ?Q .	
rdfp11	?x :sameAs ?_x . ?y :sameAs ?_y . ?x ?P ?y .⇒ ?_x ?P ?_y .	?_x $\in \mathcal{U} \cup \mathcal{B}$
rdfp12a	?C :equivalentClass ?D . ⇒ ?C rdfs:subClassOf ?D .	
rdfp12b	?C :equivalentClass ?D . ⇒ ?D rdfs:subClassOf ?C .	?D$\in \mathcal{U} \cup \mathcal{B}$
rdfp12c	?C rdfs:subClassOf ?D . ?D rdfs:subClassOf ?C . ⇒ ?C :equivalentClass ?D .	
rdfp13a	?P :equivalentProperty ?Q . ⇒ ?P rdfs:subPropertyOf ?Q .	
rdfp13b	?P :equivalentProperty ?Q . ⇒ ?Q rdfs:subPropertyOf ?P .	?Q$\in \mathcal{U} \cup \mathcal{B}$
rdfp13c	?P rdfs:subPropertyOf ?Q . ?Q rdfs:subPropertyOf ?P . ⇒ ?P :equivalentProperty ?Q .	
rdfp14a	?C :hasValue ?y ; :onProperty ?P . ?x ?P ?y . ⇒ ?x a ?C .	
rdfp14b	?C :hasValue ?y ; :onProperty ?P . ?x a ?C . ⇒ ?x ?P ?y .	?P$\in \mathcal{U} \cup \mathcal{B}$
rdfp15	?C :someValuesFrom ?D ; :onProperty ?P . ?x ?P ?y . ?y a ?D . ⇒ ?x a ?C .	
rdfp16	?C :allValuesFrom ?D ; :onProperty ?P . ?x a ?C; ?P ?y . ⇒ ?y a ?D .	?y$\in \mathcal{U} \cup \mathcal{B}$

Table 1: Ter-Horst rules from [42, 43] in Turtle-style syntax

[a] `_:bl` is a *surrogate blank node* given by an injective function on the literal `?l`

SAOR	rule	where
	$\mathcal{R}0$: only terminological patterns in antecedent	
rdfc0	?C :oneOf (?x$_1$ ... ?x$_n$) . ⇒ ?x$_1$ ... ?x$_n$ a ?C .	?C $\in \mathcal{B}$
	$\mathcal{R}1$: at least one terminological/only one assertional pattern in antecedent	
rdfs2	**?P** rdfs:domain ?C . ?x ?P ?y . ⇒ ?x a ?C .	
rdfs3′	**?P** rdfs:range ?C . ?x ?P ?y . ⇒ ?y a ?C .	
rdfs7′	**?P** rdfs:subPropertyOf ?Q . ?x ?P ?y . ⇒ ?x ?Q ?y .	
rdfs9	**?C** rdfs:subClassOf ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfp3′	**?P** a :SymmetricProperty . ?x ?P ?y . ⇒ ?y ?P ?x .	
rdfp8a′	**?P** :inverseOf ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	
rdfp8b′	?P :inverseOf **?Q** . ?x ?Q ?y . ⇒ ?y ?P ?x .	
rdfp12a′	**?C** :equivalentClass ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfp12b′	?C :equivalentClass **?D** . ?x a ?D . ⇒ ?x a ?C .	
rdfp13a′	**?P** :equivalentProperty ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	
rdfp13b′	?P :equivalentProperty **?Q** . ?x ?Q ?y . ⇒ ?y ?P ?x .	
rdfp14a′	?C :hasValue ?y ; :onProperty **?P** . ?x ?P ?y . ⇒ ?x a ?C .	?C $\in \mathcal{B}$
rdfp14b′	?C :hasValue ?y ; :onProperty ?P . ?x a ?C . ⇒ ?x ?P ?y .	?C $\in \mathcal{B}$
rdfc1	?C :unionOf (?C$_1$...**?C$_i$**...?C$_n$) . ?x a ?C$_i$[a] . ⇒ ?x a ?C .	?C $\in \mathcal{B}$
rdfc2	?C :minCardinality 1 ; :onProperty **?P** . ?x ?P ?y . ⇒ ?x a ?C .	?C $\in \mathcal{B}$
rdfc3a	?C :intersectionOf (?C$_1$ ... ?C$_n$) . ?x a ?C . ⇒ ?x a ?C$_1$, ..., ?C$_n$.	?C $\in \mathcal{B}$
rdfc3b	?C :intersectionOf (**?C$_1$**) . ?x a ?C$_1$. ⇒ ?x a ?C . [b]	?C $\in \mathcal{B}$
	$\mathcal{R}2$: at least one terminological/multiple assertional patterns in antecedent	
rdfp1′	**?P** a :FunctionalProperty . ?x ?P ?y , ?z . ⇒ ?y :sameAs ?z .	
rdfp2	**?P** a :InverseFunctionalProperty . ?x ?P ?z . ?y ?P ?z . ⇒ ?x :sameAs ?y .	
rdfp4	**?P** a :TransitiveProperty . ?x ?P ?y . ?y ?P ?z . ⇒ ?x ?P ?z .	
rdfp15′	?C :someValuesFrom **?D** ; :onProperty **?P** . ?x ?P ?y . ?y a ?D . ⇒ ?x a ?C .	?C $\in \mathcal{B}$
rdfp16′	?C :allValuesFrom ?D ; :onProperty ?P . ?x a ?C ; ?P ?y . ⇒ ?y a ?D .	?C $\in \mathcal{B}$
rdfc3c	?C :intersectionOf (**?C$_1$ ... ?C$_n$**) . ?x a ?C$_1$, ..., ?C$_n$. ⇒ ?x a ?C .	?C $\in \mathcal{B}$
rdfc4a	?C :cardinality 1 ; :onProperty ?P . ?x a ?C ; ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?C $\in \mathcal{B}$
rdfc4b	?C :maxCardinality 1 ; :onProperty ?P . ?x a ?C ; ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?C $\in \mathcal{B}$
	$\mathcal{R}3$: only assertional patterns in antecedent	
rdfp6′	?x :sameAs ?y . ⇒ ?y :sameAs ?x .	
rdfp7	?x :sameAs ?y . ?y :sameas ?z . ⇒ ?x :sameAs ?z .	
rdfp11′	?x :sameAs ?_x ; ?P ?y .⇒ ?_x ?P ?y . [c]	
rdfp11″	?y :sameAs ?_y . ?x ?P ?y .⇒ ?x ?P ?_y . [c]	

Table 2: Supported rules in Turtle-style syntax. Terminological patterns are underlined whereas assertional patterns are not; further, rules are grouped according to arity of terminological/assertional patterns in the antecedent. The source of a terminological pattern instance must speak authoritatively for at least one boldface variable binding for the rule to fire.

[a] $?C_i \in \{?C_1, ..., ?C_n\}$

[b] **rdfs3b** is a special case of **rdfs3c** with one A-Box pattern and thus falls under $\mathcal{R}1$.

[c] Only where ?p is not an RDFS/OWL property used in any of our rules (e.g., see $\mathcal{P}_{SAOR}$, Section 3.3)

pD* Rules Directly Supported From the set of pD* rules, we directly support rules **rdfs2**, **rdfs9**, **rdfp2**, **rdfp4**, **rdfp7**, and **rdfp17**.

pD* Omissions: Extended-Axiomatic Statements We avoid pD* rules which specifically produce what we term *extended-axiomatic* statements mandated by RDFS and OWL semantics. Firstly, we do not infer the set of pD* axiomatic triples, which are listed in [43, Table 3] and [43, Table 6] for RDF(S) and OWL respectively; according to pD*, these are inferred for the empty graph. Secondly, we do not materialise membership assertions for `rdfs:Resource` which would hold for every URI and blank node in a graph. Thirdly, we do not materialise reflexive `:sameAs` membership assertions, which again hold for every URI and blank node in a graph. We see such statements as inflationary and orthogonal to our aim of reduced output.

pD* Amendments: `:sameAs` Inferencing From the previous set of omissions, we do not infer reflexive `:sameAs` statements. However, such reflexive statements are required by pD* rule **rdfp11**. We thus fragment the rule into **rdfp11′** and **rdfp11″** which allows for the same inferencing without such reflexive statements.

In a related issue, we wittingly do not allow `:sameAs` inferencing to interfere with terminological data: for example, we do not allow `:sameAs` inferencing to affect properties in the predicate position of a triple or classes in the object position of an `rdf:type` triple. In [23] we showed that `:sameAs` inferencing through `:InverseFunctionalProperty` reasoning caused fallacious equalities to be asserted due to noisy web data. This is the primary motivation for us also omitting rules **rdfp9**, **rdfp10** and the reason why we place the restriction on `?p` for our rule **rdfp11″**; we do not want noisy equality inferences to be reflected in the terminological segment of our knowledge-base, nor to affect the class and property positions of membership assertions.

pD* Omissions: Terminological Inferences From pD*, we also omit rules which infer only terminological statements: namely **rdf1**, **rdfs5**, **rdfs6**, **rdfs8**, **rdfs10**, **rdfs11**, **rdfs12**, **rdfs13**, **rdfp9**, **rdfp10**, **rdfp12*** and **rdfp13***. As such, our use-case is query-answering over assertional data; we therefore focus in this paper on materialising assertional data.

We have already motivated omission of inference through `:sameAs` rules **rdfp9** and **rdfp10**. Rules **rdf1**, **rdfs8**, **rdfs12** and **rdfs13** infer memberships of, or subclass/subproperty relations to, RDF(S) classes and properties; we are not interested in these primarily syntactic statements which are not directly used in our inference rules. Rules **rdfs6** and **rdfs10** infer reflexive memberships of `rdfs:subPropertyOf` and `rdfs:subClassOf` meta-properties which are used in our inference rules; clearly however, these reflexive statements will not lead to unique assertional inferences through related rules **rdfs7′** or **rdfs9** respectively. Rules **rdfs5** and **rdfs11** infer transitive memberships again of `rdfs:subPropertyOf` and `rdfs:subClassOf`; again however, exhaustive application of rules **rdfs7′** or **rdfs9** respectively ensures that all possible assertional inferences are materialised without the need for the transitive rules. Rules **rdfp12c** and **rdfp13c** infer additional `:equivalentClass`/`:equivalentProperty` statements from `rdfs:subClassOf`/`rdfs:subPropertyOf` statements where assertional inferences can instead be conducted through two applications each of rules **rdfs9** and **rdfs7′** respectively.

pD* Amendments: Direct Assertional Inferences The observant reader may have noticed that we did not dismiss inferencing for rules **rdfp12a**,**rdfp12b**/**rdfp13a**,**rdfp13b** which translate `:equivalentClass`/`:equivalentProperty` to `rdfs:subClassOf`/ `rdfs:subPropertyOf`. In pD*, these rules are required to support indirect assertional inferences through rules **rdfs9** and **rdfs7** respectively; we instead support assertional inferences directly from the `:equivalentProperty`/`:equivalentClass` statements using symmetric rules **rdfp12a′**,**rdfp12b′**/**rdfp13a′**,**rdfp13b′**.

pD* Omissions: Existential Variables in Consequent We avoid rules with existential variables in the consequent; such rules would require adaptation of the T_r operator so as to "invent" new blank nodes for each rule application, with undesireable effects for forward-chaining reasoning regarding termination. For example, like pD*, we only support inferences in one direction for `:someValuesFrom` and avoid a rule such as:

```
?C :someValuesFrom ?D ; :onProperty ?P . ?x a ?C ⇒ ?x ?P
_:b .  _:b a ?D .
```

Exhaustive application of the rule to, for example, the following data (more generally where `?D` is a subclass of `?C`):

```
ex:Person rdfs:subClassOf [:someValuesFrom ex:Person ;
                           :onProperty ex:mother .] .
_:Tim a ex:Person .
```

would infer infinite triples of the type:

```
_:Tim ex:mother _:b0 .
_:b0 a ex:Person ; ex:mother _:b1 .
_:b1 a ex:Person ; ex:mother _:b2 .
...
```

In fact, this rule is listed in [43] as **rdf-svx** which forms an extension of pD* entailment called *pD*sv*. This rule is omitted from pD* and from SAOR due to obvious side-effects on termination and complexity.

Unlike pD*, we also avoid inventing so called "surrogate" blank nodes for the purposes of representing a literal in intermediary inferencing steps (Rules **lg**, **gl**, **rdf2-D**, **rdfs1** in RDFS/D* entailment). Thus, we also do not support datatype reasoning (Rule **rdf2-D**) which involves the creation of surrogate blank nodes. Although surrogate blank nodes are created according to a direct mapping from a finite set of literals (and thus, do not prevent termination), we view "surrogate statements" as inflationary.

pD* Amendments: Relaxing Literal Restrictions Since we do not support surrogate blank nodes as representing literals, we instead relax restrictions placed on pD* rules. In pD*, blank nodes are allowed in the predicate position of triples; however, the restriction on literals in the subject and predicate position still applies: literals are restricted from travelling to the subject or predicate position of a consequent (see *where* column, Table 1). Thus, surrogate blank nodes are required in pD* to represent literals in positions where they would otherwise not be allowed.

We take a different approach whereby we allow literals directly in the subject and predicate position for intermediate inferences. Following from this, we remove pD* literal restrictions on rules **rdfs3**, **rdfs7**, **rdfp1**, **rdfp3**, **rdfp6**, **rdfp8***, **rdfp14b**, **rdfp16** for intermediate inferences and omit any inferred non-RDF statements from being written to the final output.

Additions to pD* In addition to pD*, we also include some "class based entailment" from OWL, which we call C-entailment. We name such rules using the **rdfc*** stem, following the convention from P-entailment. We provide limited support for enumerated classes (**rdfc0**), union class descriptions (**rdfc1**), intersection class descriptions (**rdfc3***)[6], as well as limited cardinality constraints (**rdfc2**, **rdfc4***).

pD* Amendments: Enforcing OWL Abstract Syntax Restrictions Finally, unlike pD*, we enforce blank nodes as mandated by the OWL Abstract Syntax [37], wherein certain abstract syntax constructs (most importantly in our case: *unionOf(description*$_1$ *... description*$_n$*)*, *intersectionOf(description*$_1$*...description*$_n$*)*, *oneOf(iID*$_1$*...iID*$_n$*)*, *restriction(ID allValuesFrom(range))*, *restriction(ID someValuesFrom(required))*, *restriction(ID value(value))*, *restriction(ID maxCardinality(max))*, *restriction(ID minCardinality(min))*, *restriction(ID cardinality(card))* and *SEQ item*$_1$*...item*$_n$) are strictly mapped to RDF triples with blank nodes enforced for certain positions: such mapping is necessitated by the idiosyncrasies of representing OWL in RDF. Although the use of URIs in such circumstances is allowed by RDF, we enforce the use of blank nodes for terminological patterns in our ruleset; to justify, let us look at the following problematic example of OWL triples taken from two sources:

Example 20 :
# FROM SOURCE `<ex:>`
```
ex:Person :onProperty ex:parent ; :someValuesFrom ex:Person .
```
# FROM SOURCE `<ex2:>`
```
ex:Person :allValuesFrom ex2:Human .
```
◇

According to the abstract syntax mapping, neither of the restrictions should be identified by a URI (if blank nodes were used instead of `ex:Person` as mandated by the abstract syntax, such a problem could not occur as each web-graph is given a unique set of blank nodes). If we consider the RDF-merge of the two graphs, we will be unable to distinguish which restriction the `:onProperty` value should be applied to. As above, allowing URIs in these positions would enable "syntactic interference" between data sources. Thus, in our ruleset, we always enforce blank-nodes as mandated by the OWL abstract syntax; this specifically applies to pD* rules **rdfp14***, **rdfp15** and **rdfp16** and to all of our C-entailment rules **rdfc***. We denote the restrictions in the **where** column of Table 2. Indeed, in our treatment of terminological collection statements, we enforced blank nodes in the subject position of `rdf:first`/`rdf:rest` membership assertions, as well as blank nodes in the object position of non-terminating `rdf:rest` statements; these are analogously part of the OWL abstract syntax restrictions.

3.3 Separation of T-Box from A-Box

Aside from the differences already introduced, our primary divergence from the pD* fragment and traditional rule-based approaches is that we separate terminological data

[6]In [43], rules using RDF collection constructs were not included (such as our rules **rdfc0**,**rdfc1**,**rdfc3***) as they have variable antecedent-body length and, thus, can affect complexity considerations. It was informally stated that `:intersectionOf` and `:unionOf` could be supported under pD* through reduction into subclass relations; however no rules were explicitly defined and our rule **rdfc3b** could not be supported in this fashion. We support such rules here since we are not so concerned for the moment with theoretical worst-case complexity, but are more concerned with the practicalities of web reasoning.

from assertional data according to their use of the RDF(S) and OWL vocabulary; these are commonly known as the "T-Box" and "A-Box" respectively (loosely borrowing Description Logics terminology). In particular, we require a separation of T-Box data as part of a core optimisation of our approach; we wish to perform a once-off load of T-Box data from our input knowledge-base into main memory.

Let $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$, resp., be the exact set of RDF(S)/OWL meta-properties and -classes used in our inference rules; viz. $\mathcal{P}_{SAOR} = \{$ `rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf, :allValuesFrom, :cardinality, :equivalentClass, :equivalentProperty, :hasValue, :intersectionOf, :inverseOf, :maxCardinality, :minCardinality, :oneOf, :onProperty, :sameAs, :someValuesFrom, :unionOf` $\}$ and, resp., $\mathcal{C}_{SAOR} = \{$ `:FunctionalProperty, :InverseFunctionalProperty, :SymmetricProperty, :TransitiveProperty` $\}$; our T-Box is a set of terminological triples restricted to only include membership assertions for $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$ and the set of terminological collection statements. Table 2 identifies T-Box patterns by underlining. Statements from the input knowledge-base that match these patterns are all of the T-Box statements we consider in our reasoning process: inferred statements or statements that do not match one of these patterns are not considered being part of the T-Box, but are treated purely as assertional. We now define our T-Box:

Definition 24 (T-Box) *Let $\mathcal{T}_{\mathcal{G}}$ be the union of all graph pattern instances from a graph $\mathcal{G}$ for a terminological (underlined) graph pattern in Table 2; i.e., $\mathcal{T}_{\mathcal{G}}$ is itself a graph. We call $\mathcal{T}_{\mathcal{G}}$ the* T-Box of $\mathcal{G}$.

Also, let $\mathcal{P}^{domP}_{SAOR} = \{$ `rdfs:domain, rdfs:range, rdfs:subPropertyOf, :equivalentProperty, :inverseOf` $\}$ and $\mathcal{P}^{ranP}_{SAOR} = \{$ `rdfs:subPropertyOf, :equivalentProperty, :inverseOf, :onProperty` $\}$, We call ϕ a *property in T-Box $\mathcal{T}$* if there exists a triple $t \in \mathcal{T}$ where

- $s = \phi$ and $p \in \mathcal{P}^{domP}_{SAOR}$
- $p \in \mathcal{P}^{ranP}_{SAOR}$ and $o = \phi$
- $s = \phi$, p=`rdf:type` and $o \in \mathcal{C}_{SAOR}$

Similarly, let $\mathcal{P}^{domC}_{SAOR} = \{$ `rdfs:subClassOf, :allValuesFrom, :cardinality, :equivalentClass, :hasValue, :intersectionOf, :maxCardinality, :minCardinality, :oneOf, :onProperty, :someValuesFrom, :unionOf` $\}$, $\mathcal{P}^{ranC}_{SAOR} = \{$ `rdf:first, rdfs:domain, rdfs:range, rdfs:subClassOf, :allValuesFrom, :equivalentClass, :someValuesFrom` $\}$. We call χ a *class in T-Box $\mathcal{T}$* if there exists a triple $t \in \mathcal{T}$ where

- $p \in \mathcal{P}^{domC}_{SAOR}$ and $s = \chi$
- $p \in \mathcal{P}^{ranC}_{SAOR}$ and $o = \chi$

We define the signature of a T-Box $\mathcal{T}$ to be the set of all properties and classes in $\mathcal{T}$ as above, which we denote by $sig(\mathcal{T})$.

For our knowledge-base $\mathbb{KB}$, we define our *T-Box* $\mathbb{T}$ as the set of all pairs $(\mathcal{T}_{\mathcal{W}'}, c)$ where $(\mathcal{W}', c) \in \mathbb{KB}$ and $\mathcal{T}_{\mathcal{W}'} \neq \emptyset$. Again, we may use the intuitive notation $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. We define our A-Box $\mathbb{A}$ as containing all of the statements in $\mathbb{KB}$, including $\mathbb{T}$

and the set of class and property membership assertions possibly using identifiers in $\mathcal{P}_{SAOR} \cup \mathcal{C}_{SAOR}$; i.e., unlike description logics, our $\mathbb{A}$ is synonymous with our $\mathbb{KB}$. We use the term A-Box to distinguish data that are stored on-disk (which includes T-Box data also stored in memory).

We now define our notion of a *$\mathcal{T}$-split inference rule*, whereby part of the antecedent is a basic graph pattern strictly instantiated by a static T-Box $\mathcal{T}$.

Definition 25 ($\mathcal{T}$-split inference rule) *We define a $\mathcal{T}$-*split inference rule *as a triple ($\mathcal{A}nte_{\mathcal{T}}, \mathcal{A}nte_{\mathcal{G}}, \mathcal{C}on$), where $\mathcal{A}nte_{\mathcal{T}}$ is a basic graph pattern matched by a static T-Box $\mathcal{T}$ and $\mathcal{A}nte_{\mathcal{G}}$ is matched by data in the graph $\mathcal{G}$, $\mathcal{C}on$ does not contain blank nodes, $\mathcal{V}(\mathcal{C}on) \neq \emptyset$, $\mathcal{V}(\mathcal{C}on) \subseteq \mathcal{V}(\mathcal{A}nte_{\mathcal{T}}) \cup \mathcal{V}(\mathcal{A}nte_{\mathcal{G}})$; also, if both $\mathcal{A}nte_{\mathcal{T}}$ and $\mathcal{A}nte_{\mathcal{G}}$ are non-empty, then $\mathcal{V}(\mathcal{A}nte_{\mathcal{T}}) \cap \mathcal{V}(\mathcal{A}nte_{\mathcal{G}}) \neq \emptyset$*

We generally write $(\mathcal{A}nte_{\mathcal{T}}, \mathcal{A}nte_{\mathcal{G}}, \mathcal{C}on)$ as $\underline{\mathcal{A}nte_{\mathcal{T}}}\mathcal{A}nte_{\mathcal{G}} \Rightarrow \mathcal{C}on$ where Table 2 follows this convention. We call $\mathcal{A}nte_{\mathcal{T}}$ the terminological or T-Box antecedent pattern and $\mathcal{A}nte_{\mathcal{G}}$ the assertional or A-Box antecedent pattern.

We now define three disjoint sets of $\mathcal{T}$-split rules which consist of only a T-Box graph pattern, both a T-Box and A-Box graph pattern and only an A-Box graph pattern:

Definition 26 (Rule-sets $\mathcal{R}_{\mathcal{T}}, \mathcal{R}_{\mathcal{T}\mathcal{G}}, \mathcal{R}_{\mathcal{G}}$) *We define $\mathcal{R}_{\mathcal{T}}$ as the set of $\mathcal{T}$-split rules for which $\mathcal{A}nte_{\mathcal{T}} \neq \emptyset$ and $\mathcal{A}nte_{\mathcal{G}} = \emptyset$. We define $\mathcal{R}_{\mathcal{T}\mathcal{G}}$ as the set of $\mathcal{T}$-split rules for which $\mathcal{A}nte_{\mathcal{T}} \neq \emptyset$ and $\mathcal{A}nte_{\mathcal{G}} \neq \emptyset$. We define $\mathcal{R}_{\mathcal{G}}$ as the set of $\mathcal{T}$-split rules for which $\mathcal{A}nte_{\mathcal{T}} = \emptyset$ and $\mathcal{A}nte_{\mathcal{G}} \neq \emptyset$.*

In Table 2, we categorise the $\mathcal{T}$-split rules into four rulesets: $\mathcal{R}0 \subset \mathcal{R}_{\mathcal{T}}$; $\mathcal{R}1 \subset \mathcal{R}_{\mathcal{T}\mathcal{G}}$ where $|\mathcal{A}nte_{\mathcal{G}}| = 1$; $\mathcal{R}2 \subset \mathcal{R}_{\mathcal{T}\mathcal{G}}$ where $|\mathcal{A}nte_{\mathcal{G}}| > 1$ and $\mathcal{R}0 \subset \mathcal{R}_{\mathcal{G}}$. We now introduce the notion of a $\mathcal{T}$-split inference rule application for a graph $\mathcal{G}$ w.r.t. a T-Box $\mathcal{T}$:

Definition 27 ($\mathcal{T}$-split inference rule application) *We define a $\mathcal{T}$-*split rule application *to be $T_r(\mathcal{T}, \mathcal{G})$ for $r = (\mathcal{A}nte_{\mathcal{T}}, \mathcal{A}nte_{\mathcal{G}}, \mathcal{C}on)$ as follows:*

$$T_r(\mathcal{T}, \mathcal{G}) = \{\mu(\mathcal{C}on) \mid \exists \mu \text{ such that } \mu(\mathcal{A}nte_{\mathcal{T}}) \subseteq \mathcal{T} \text{ and } \mu(\mathcal{A}nte_{\mathcal{G}}) \subseteq \mathcal{G}\}$$

Again, $T_{\mathcal{R}}(\mathcal{T}, \mathcal{G}) = \bigcup_{r \in \mathcal{R}} T_r(\mathcal{T}, \mathcal{G})$; also, given $\mathcal{T}$ as static, the exhaustive application of the $T_{\mathcal{R}}(\mathcal{T}, \mathcal{G})$ up to the least fixpoint is called the *$\mathcal{T}$-split closure* of $\mathcal{G}$, denoted as $Cl_{\mathcal{R}}(\mathcal{T}, \mathcal{G})$. Again we use abbreviations such as $T_R(\mathbb{T}, \mathbb{KB})$ and $Cl_{\mathcal{R}}(\mathbb{T}, \mathbb{KB})$, where $\mathbb{KB}$ should be interpreted as $\bigcup_{\mathcal{W}' \in \mathbb{KB}} \mathcal{W}'$ and $\mathbb{T}$ as $\bigcup_{\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}} \mathcal{T}_{\mathcal{W}'}$.

Please note that since we enforce blank nodes in all positions mandated by the OWL abstract syntax for our rules, each instance of a given graph pattern $\mathcal{A}nte_{\mathcal{T}}$ can only contain triples from one web graph $\mathcal{W}'$ where $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. Let $\mathcal{V}_{\mathcal{B}}(\mathcal{GP})$ be the set of all variables in a graph pattern $\mathcal{GP}$ which we restrict to only be instantiated by a blank node according to the abstract syntax. For all $\mathcal{A}nte_{\mathcal{T}}$ in our rules where $|\mathcal{A}nte_{\mathcal{T}}| > 1$ let $\mathcal{A}nte_{\mathcal{T}}^{-}$ be any proper non-empty subset of $\mathcal{A}nte_{\mathcal{T}}$; we can then say that $\mathcal{V}_{\mathcal{B}}(\mathcal{A}nte_{\mathcal{T}}^{-}) \cap \mathcal{V}_{\mathcal{B}}(\mathcal{A}nte_{\mathcal{T}} \setminus \mathcal{A}nte_{\mathcal{T}}^{-}) \neq \emptyset$. In other words, since for every rule either (i) $\mathcal{A}nte_{\mathcal{T}} = \emptyset$; or (ii) $\mathcal{A}nte_{\mathcal{T}}$ consists of a single triple pattern; or otherwise (iii) no sub-pattern of $\mathcal{A}nte_{\mathcal{T}}$ contains a unique set of blank-node enforced variables; then a given instance of $\mathcal{A}nte_{\mathcal{T}}$ can only contain triples from one web-graph with unique blank nodes as is enforced by our knowledge-base. For our ruleset, we can then say that $T_R(\mathbb{T}, \mathbb{KB}) = T_R(\bigcup_{\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}} \mathcal{T}_{\mathcal{W}'}, \mathbb{KB}) = \bigcup_{\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}} T_R(\mathcal{T}_{\mathcal{W}'}, \mathbb{KB})$. In other words, one web-graph cannot re-use structural statements in another web-graph to instantiate a T-Box pattern in our rule; this has bearing on our notion of authoritative reasoning which will be highlighted at the end of Section 3.4.

Further, a separate static T-Box within which inferences are not reflected has implications upon the completeness of reasoning w.r.t. the presented ruleset. Although, as presented in Section 3.2, we do not infer terminological statements and thus can support most inferences directly from our static T-Box, SAOR still does not fully support meta-modelling [33]: by separating the T-Box segment of the knowledge-base, we do not support all possible entailments from the simultaneous description of both a class (or property) and an indvidual. In other words, we do not fully support inferencing for meta-classes or meta-properties defined outside of the RDF(S)/OWL specification.

However, we do provide limited reasoning support for meta-modelling in the spirit of "punning" by conceptually separating the individual-, class- or property-meanings of a resource (c.f. [14]). More precisely, during reasoning we not only store the T-Box data in memory, but also store the data on-disk in the A-Box. Thus, we perform punning in one direction: viewing class and property descriptions which form our T-Box also as individuals. Interestingly, although we do not support terminological reasoning directly, we can through our limited punning perform reasoning for terminological data based on the RDFS descriptions provided for the RDFS and OWL specifications. For example, we would infer the following by storing the three input statements in both the T-Box and the A-Box:

```
rdfs:subClassOf rdfs:domain rdfs:Class; rdfs:range rdfs:Class
.
ex:Class1 rdfs:subClassOf ex:Class2 .  ⇒
ex:Class1 a rdfs:Class .  ex:Class2 a rdfs:Class .
```

However, again our support for meta-modelling is limited; SAOR does not fully support so-called "non-standard usage" of RDF(S) and OWL: the use of properties and classes which make up the RDF(S) and OWL vocabularies in locations where they have not been intended, cf. [6, 34]. We adapt and refine the definition of non-standard vocabulary use for our purposes according to the parts of the RDF(S) and OWL vocabularies relevant for our inference ruleset:

Definition 28 (Non-Standard Vocabulary Usage) *An RDF triple t has* non-standard vocabulary usage *if one of the following conditions holds:*

1. *a property in $\mathcal{P}_{SAOR}$ appears in a position different from the predicate position.*
2. *a class in $\mathcal{C}_{SAOR}$ appears in a position different from the object position of an* `rdf:type` *triple.*

Continuing, we now introduce the following example wherein the first input statement is a case of non-standard usage with `rdfs:subClassOf`$\in \mathcal{P}_{SAOR}$ in the object position:[7]

```
ex:subClassOf rdfs:subPropertyOf rdfs:subClassOf .
ex:Class1 ex:subClassOf ex:Class2 .  ⇒
ex:Class1 rdfs:subClassOf ex:Class2 .
```

[7]A similar example from the Web can be found at `http://thesauri.cs.vu.nl/wordnet/rdfs/wordnet2b.owl`.

We can see that SAOR provides inference through `rdfs:subPropertyOf` as per usual; however, the inferred triple will not be reflected in the T-Box, thus we are incomplete and will not translate members of `ex:Class1` into `ex:Class2`. As such, non-standard usage may result in T-Box statements being produced which, according to our limited form of punning, will not be reflected in the T-Box and will lead to incomplete inference.

Indeed, there may be good reason for not fully supporting non-standard usage of the ontology vocabulary: non-standard use could have unpredictable results even under our simple rule-based entailment if we were to fully support meta-modelling. One may consider a finite combination of only four non-standard triples that, upon naive reasoning, would explode all web resources R by inferring $|R|^3$ triples, namely:

```
rdfs:subClassOf rdfs:subPropertyOf rdfs:Resource.
rdfs:subClassOf rdfs:subPropertyOf rdfs:subPropertyOf.
rdf:type rdfs:subPropertyOf rdfs:subClassOf.
rdfs:subClassOf rdf:type :SymmetricProperty.
```

The exhaustive application of standard RDFS inference rules plus inference rules for property symmetry together with the inference for class membership in `rdfs:Resource` for all collected resources in typical rulesets such as pD* lead to inference of any possible triple $(r_1\ r_2\ r_3)$ for arbitrary $r_1, r_2, r_3 \in R$.

Thus, although by maintaining a separate static T-Box we are incomplete w.r.t non-standard usage, we show that complete support of such usage of the RDFS/OWL vocabularies is undesirable for the Web.[8]

3.4 Authoritative Reasoning against Ontology Hijacking

During initial evaluation of a system which implements reasoning upon the above ruleset, we encountered a behaviour which we term "ontology hijacking", symptomised by a perplexing explosion of materialised statements. For example, we noticed that for a single `foaf:Person` membership assertion, SAOR inferred in the order of hundreds of materialised statements as opposed to the expected six. Such an explosion of statements is orthogonal to the aim of reduced materialised statements we have outlined for SAOR; thus, SAOR is designed to annul the diagnosed problem of ontology hijacking through anaylsis of the authority of web sources for T-Box data. Before formally defining ontology hijacking and our proposed solution, let us give some preliminary definitions:

Definition 29 (Authoritative Source) *A web-graph $\mathcal{W}$ from source (context) c speaks authoritatively about an RDF term n iff:*

1. *$n \in \mathcal{B}$; or*
2. *$n \in \mathcal{U}$ and c coincides with, or is redirected to by, the namespace[9] of n.*

[8]In any case, as we will see in Section 3.4, our application of authoritative analysis would not allow such arbitrary third-party re-definition of core RDF(S)/OWL constructs.

[9]Here, slightly abusing XML terminology, by "namespace" of a URI we mean the prefix of the URI obtained from stripping off the final NCname.

Firstly, all graphs are authoritative for blank nodes defined in that graph (remember that according to the definition of our knowledge-base, all blank nodes are unique to a given graph). Secondly, we support namespace redirects so as to conform to best practices as currently adopted by web ontology publishers.[10]

For example, as taken from the Web:

- Source `http://usefulinc.com/ns/doap` is authoritative for all classes and properties which are within the `http://usefulinc.com/ns/doap` namespace; e.g., `http://usefulinc.com/ns/doap#Project`.

- Source `http://xmlns.com/foaf/spec/` is authoritative for all classes and properties which are within the `http://xmlns.com/foaf/0.1/` namespace; e.g., `http://xmlns.com/foaf/0.1/knows`; since the property `http://xmlns.com/foaf/0.1/knows` redirects to `http://xmlns.com/foaf/spec/`.

We consider the authority of sources speaking about classes and properties in our T-Box to counter-act ontology hijacking; ontology hijacking is the assertion of a set of non-authoritative T-Box statements such that could satisfy the T-Box antecedent pattern of a rule in $\mathcal{R}_{TG}$ (i.e., those rules with at least one terminological and at least one assertional triple pattern in the antecedent). Such third-party sources can then cause arbitrary inferences on membership assertions of classes or properties (contained in the A-Box) for which they speak non-authoritatively. We can say that only rules in $\mathcal{R}_{TG}$ are relevant to ontology hijacking since: (i) inferencing on $\mathcal{R}_G$, which does not contain any T-Box patterns, cannot be affected by non-authoritative T-Box statements; and (ii) the $\mathcal{R}_T$ ruleset does not contain any A-Box antecedent patterns and therefore, cannot directly hijack assertional data (i.e., in our scenario, the `:oneOf` construct can be viewed as directly asserting memberships, and is unable, according to our limited support, to directly redefine sets of individuals). We now define ontology hijacking:

Definition 30 (Ontology Hijacking) *Let $\mathcal{T}_{\mathcal{W}}$ be the T-Box extracted from a web-graph $\mathcal{W}$ and let $\widehat{sig}(\mathcal{W})$ be the set of classes and properties for which $\mathcal{W}$ speaks authoritatively; then if $Cl_{\mathcal{R}_{\mathcal{T}\mathcal{G}}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G}) \neq \mathcal{G}$ for any $\mathcal{G}$ not mentioning any element of $\widehat{sig}(\mathcal{W})$, we say that web-graph $\mathcal{W}$ is performing* ontology hijacking.

In other words, ontology hijacking is the contribution of statements about classes and/or properties in a non-authoritative source such that reasoning on those classes and/or properties is affected. One particular method of ontology hijacking is defining new super-classes or properties of third-party classes or properties.

As a concrete example, if one were to publish today a description of a property in an ontology (in a location non-authoritative for `foaf:` but authoritative for `my:`), `my:name`, within which the following was stated: `foaf:name rdfs:subPropertyOf my:name .`, that person would be hijacking the `foaf:name` property and effecting the translation of all `foaf:name` statements in the web knowledge-base into `my:name` statements as well.

However, if the statement were `my:name rdfs:subPropertyOf foaf:name.` instead, this would not constitute a case of ontology hijacking but would be a valid example of translating from a local authoritative property into an external non-authoritative property.

[10]See Appendix A&B of `http://www.w3.org/TR/swbp-vocab-pub/`

Ontology hijacking is problematic in that it vastly increases the amount of statements that are materialised and can potentially harm inferencing on data contributed by other parties. With respect to materialisation, the former issue becomes prominent: members of classes/properties from popular/core ontologies get translated into a plethora of conceptual models described in obscure ontologies; we quantify the problem in Section 5. However, taking precautions against harmful ontology hijacking is growing more and more important as the Semantic Web features more and more attention; motivation for spamming and other malicious activity propagates amongst certain parties with ontology hijacking being a prospective avenue. With this in mind, we assign sole responsibility for classes and properties and reasoning upon their members to those who maintain the authoritative specification.

Related to the idea of ontology hijacking is the idea of "non-conservative extension" described in the Description Logics literature: cf. [13, 31, 27]. However, the notion of a "conservative extension" was defined with a slightly different objective in mind: according to the notion of deductively conservative extensions, a graph G_a is only considered malicious towards G_b if it causes additional inferences with respect to the intersection of the signature of the original G_b with the newly inferred statements. Returning to the former `my:name` example from above, defining a super-property of `foaf:name` would still constitute a conservative extension: the closure without the non-authoritative `foaf:name rdfs:subPropertyOf my:name .` statement is the same as the closure with the statement after all of the `my:name` membership assertions are removed. However, further stating that `my:name a :InverseFunctionalPro-perty.` would not satisfy a model conservative extension since members of `my:name` might then cause equalities in other remote ontologies as side-effects, independent from the newly defined signature. Summarising, we can state that every non-conservative extension (with respect to our notion of deductive closure) constitutes a case of ontology hijacking, but not vice versa; non-conservative extension can be considered "harmful" hijacking whereas the remainder of ontology hijacking cases can be considered "inflationary".

To negate ontology hijacking, we only allow inferences through *authoritative rule applications*, which we now define:

Definition 31 (Authoritative Rule Application) *Again let $\widehat{sig}(\mathcal{W})$ be the set of classes and properties for which $\mathcal{W}$ speaks authoritatively and let $\mathcal{T}_{\mathcal{W}}$ be the T-Box of $\mathcal{W}$. We define an* authoritative rule application *for a graph $\mathcal{G}$ w.r.t. the T-Box $\mathcal{T}_{\mathcal{W}}$ to be a $\mathcal{T}$-split rule application $T_r(\mathcal{T}_{\mathcal{W}}, \mathcal{G})$ where additionally, if both $\mathcal{A}nte_{\mathcal{T}}$ and $\mathcal{A}nte_{\mathcal{G}}$ are non-empty ($r \in \mathcal{R}_{\mathcal{T}\mathcal{G}}$), then for the mapping μ of $T_r(\mathcal{T}_{\mathcal{W}}, \mathcal{G})$ there must exist a variable $v \in (\mathcal{V}(\mathcal{A}nte_{\mathcal{T}}) \cap \mathcal{V}(\mathcal{A}nte_{\mathcal{G}}))$ such that $\mu(v) \in \widehat{sig}(\mathcal{W})$. We denote an* authoritative rule application *by $T_{\widehat{r}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G})$.*

In other words, an authoritative rule application will only occur if the rule consists of only assertional patterns ($\mathcal{R}_{\mathcal{G}}$); or the rules consists of only terminological patterns ($\mathcal{R}_{\mathcal{T}}$); or if in application of the rule, the terminological pattern instance is from a web-graph authoritative for at least one class or property in the assertional pattern instance. The $T_{\widehat{\mathcal{R}}}$ operator follows naturally as before for a set of authoritative rules $\widehat{\mathcal{R}}$, as does the notion of authoritative closure which we denote by $Cl_{\widehat{\mathcal{R}}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G})$. We may also refer to, e.g., $T_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB})$ and $Cl_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB})$ as before for a $\mathcal{T}$-split rule application.

Table 2 identifies the authoritative restrictions we place on our rules wherein the underlined T-Box pattern is matched by a set of triples from a web-graph $\mathcal{W}$ iff $\mathcal{W}$

speaks authoritatively for at least one element matching a boldface variable in Table 2; i.e., again, for each rule, at least one of the classes or properties matched by the A-Box pattern of the antecedent must be authoritatively spoken for by an instance of the T-Box pattern. These restrictions only apply to $\mathcal{R}1$ and $\mathcal{R}2$ (which are both a subset of $\mathcal{R}_{\mathcal{TG}}$). Please note that, for example in rule **rdfp14b′** where there are no boldface variables, the variables enforced to be instantied by blank nodes will always be authoritatively spoken for: a web-graph is always authoritative for its blank nodes.

We now make the following proposition relating to the prevention of ontology-hijacking through authoritative rule application:

Proposition 27 *Given a T-Box $\mathcal{T}_{\mathcal{W}}$ extracted from a web-graph $\mathcal{W}$ and any graph $\mathcal{G}$ not mentioning any element of $\widehat{sig}(\mathcal{W})$, then $Cl_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G}) = \mathcal{G}$.*

Proof: Informally, our proposition is that the authoritative closure of a graph $\mathcal{G}$ w.r.t. some T-Box $\mathcal{T}_{\mathcal{W}}$ will not contain any inferences which constitute ontology hijacking, defined in terms of ruleset $\mathcal{R}_{\mathcal{TG}}$. Firstly, from Definition 26, for each rule $r \in \mathcal{R}_{\mathcal{TG}}$, $\mathcal{A}nte_{\mathcal{T}} \neq \emptyset$ and $\mathcal{A}nte_{\mathcal{G}} \neq \emptyset$. Therefore, from Definitions 27 & 31, for an authoritative rule application to occur for any such r, there must exist (i) a mapping μ such that $\mu(\mathcal{A}nte_{\mathcal{T}},) \subseteq \mathcal{T}_{\mathcal{W}}$ and $\mu(\mathcal{A}nte_{\mathcal{G}}) \subseteq \mathcal{G}$; and (ii) a variable $v \in (\mathcal{V}(\mathcal{A}nte_{\mathcal{T}}) \cap \mathcal{V}(\mathcal{A}nte_{\mathcal{G}}))$ such that $\mu(v) \in \widehat{sig}(\mathcal{W})$. However, since $\mathcal{G}$ does not mention any element of $\widehat{sig}(\mathcal{W})$, then there is no such mapping μ where $\mu(v) \in \widehat{sig}(\mathcal{W})$ for $v \in \mathcal{V}(\mathcal{A}nte_{\mathcal{G}})$, and $\mu(\mathcal{A}nte_{\mathcal{G}}) \subseteq \mathcal{G}$. Hence, for $r \in \mathcal{R}_{\mathcal{TG}}$, no such application $T_{\widehat{r}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G})$ will occur; it then follows that $T_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G}) = \emptyset$ and $Cl_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_{\mathcal{W}}, \mathcal{G}) = \mathcal{G}$. □

The above proposition and proof holds for a given web-graph $\mathcal{W}$; however, given a set of web-graphs where an instance of $\mathcal{A}nte_{\mathcal{T}}$ can consist of triples from more that one graph, it is possible for ontology hijacking to occur whereby some triples in the instance come from a non-authoritative graph and some from an authoritative graph. To illustrate we refer to Example 20, wherein (and without enforcing abstract syntax blank nodes) the second source could cause ontology hijacking by interfering with the authoritative definition of the class restriction in the first source as follows:

Example 21 :

# RULE (adapted so that `?C` need not be a blank node)

?C `:allValuesFrom ?D ; :onProperty` **?P** `. ?x a ?C ; ?P ?y . ⇒ ?y a ?D .`

# FROM SOURCE `<ex:>`

ex:Person `:onProperty` **ex:parent** `.`

# FROM SOURCE `<ex2:>`

`ex:Person :allValuesFrom` **ex2:Human** `.`

# ASSERTIONAL

`_:Jim a ex:Person ; ex:parent _:Jill .`

`⇒ _:Jill a ex2:Human .` ◇

Here, the above inference is authoritative according to our definition since the instance of $\mathcal{A}nte_{\mathcal{T}}$ (specifically the first statement from `source <ex:>`) speaks authoritatively for a class/property in the assertional data; however, the statement from `source <ex2:>` is causing inferences on assertional data not containing a class or property for which `source <ex2:>` is authoritative .

As previously discussed, for our ruleset, we enforce the OWL abstract syntax and thus we enforce that $\mu(\mathcal{A}nte_{\mathcal{T}}) \subseteq \mathcal{T}_{\mathcal{W}'}$ where $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. However, where this condition does not hold (i.e., an instance of $\mathcal{A}nte_{\mathcal{T}}$ can comprise of data from more than one graph), then an authoritative rule application should only occur if each web-graph contributing to an instance of $\mathcal{A}nte_{\mathcal{T}}$ speaks authoritatively for at least one class/property in the $\mathcal{A}nte_{\mathcal{G}}$ instance.

4 Reasoning Algorithm

In the following we first present observations on web data that influenced the design of the SAOR algorithm, then give an overview of the algorithm, and next discuss details of how we handle T-Box information, perform statement-wise reasoning, and deal with equality for individuals.

4.1 Characteristics of Web Data

Our algorithm is intended to operate over a web knowledge-base as retrieved by means of a web crawl; therefore, the design of our algorithm is motivated by observations on our web dataset:

1. Reasoning accesses a large slice of data in the index: we found that approximately 61% of statements in the 147m dataset and 90% in the 1.1b dataset produced inferred statements through authoritative reasoning.

2. Relative to assertional data, the volume of terminological data on the Web is small: <0.9% of the statements in the 1.1b dataset and <1.7% of statements in the 147m dataset were classifiable as SAOR T-Box statements[11].

3. The T-Box is the most frequently accessed segment of the knowledge-base for reasoning: although relatively small, all but the rules in $\mathcal{R}3$ require access to T-Box information.

Following from the first observation, we employ a file-scan batch-processing approach so as to enable sequential access over the data and avoid disk-lookups and dynamic data structures which would not perform well given high disk latency; also we avoid probing the same statements repeatedly for different rules at the low cost of scanning a given percentage of statements not useful for reasoning.

Following from the second and third observations, we optimise by placing T-Box data in a separate data structure accessible by the reasoning engine.

Currently, we hold all T-Box data in-memory, but the algorithm can be generalised to provide for a caching on-disk structure or a distributed in-memory structure as needs require.[12]

To be able to scale, we try to minimise the amount of main memory needed, given that main memory is relatively expensive and that disk-based algorithms are thus more economical [29]. Given high disk latency, we avoid using random-access on-disk data

[11]Includes some RDF collection fragments which may not be part of a class description

[12]We expect that a caching on-disk index would work well considering the distribution of membership assertions for classes and properties in web data; there would be a high hit-rate for the cache.

structures. In our previous work, a disk-based updateable random-access data structure (a B+-Tree) proved to be the bottleneck for reasoning due to a high volume of inserts, leading to frequent index reorganisations and hence inadequate performance. As a result, our algorithms are now build upon two disk-based primitives known to scale: file scanning and sorting.

4.2 Algorithm Overview

The SAOR algorithm performs a fixpoint computation by iteratively applying the rules in Table 2. Figure 1 outlines the architecture. The reasoning process can be roughly divided into the following steps:

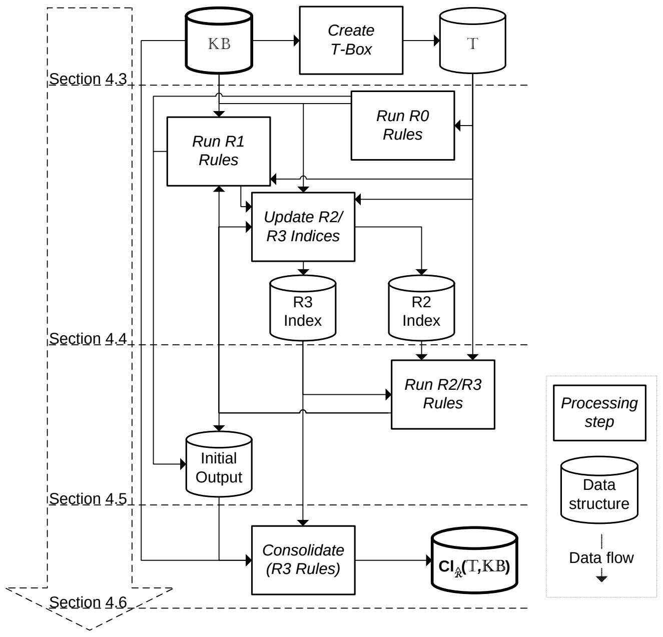


Figure 1: High-level architecture

1. Separate $\mathbb{T}$ from $\mathbb{KB}$, build in-memory representation $\mathbb{T}$, and apply ruleset $\mathcal{R}0$ (Section 4.3).

2. Perform reasoning over $\mathbb{KB}$ in a statement-wise manner (Section 4.4):

   - Execute rules with only a single A-Box triple pattern in the antecedent ($\mathcal{R}1$): join A-Box pattern with in-memory T-Box; recursively execute steps over inferred statements; write inferred RDF statements to output file.
   - Write on-disk files for computation of rules with multiple A-Box triple patterns in the antecedent ($\mathcal{R}2$); when a statement matches one of the A-Box triple patterns for these rules and the necessary T-Box join exists, the statement is written to the on-disk file for later rule computation.

- Write on-disk equality file for rules which involve equality reasoning ($\mathcal{R}3$); `:sameAs` statements found during the scan are written to an on-disk file for later computation.

3. Execute ruleset $\mathcal{R}2 \cup \mathcal{R}3$: on-disk files containing partial A-Box antecedent matches for rules in $\mathcal{R}2$ and $\mathcal{R}3$ are sequentially analysed producing further inferred statements. Newly inferred statements are again subject to step 2 above; fresh statements can still be written to on-disk files and so the process is iterative until no new statements are found (Section 4.5).

4. Finally, consolidate source data along with inferred statements according to `:sameAs` computation ($\mathcal{R}3$) and write to final output (Section 4.6).

In the following sections, we discuss the individual components and processes in the architecture as highlighted, whereafter, in Section 4.7 we show how these elements are combined to achieve closure.

4.3 Handling Terminological Data

In the following, we describe how to separate the T-Box data and how to create the data structures for representing the T-Box.

T-Box data from RDFS and OWL specifications can be acquired either from conventional crawling techniques, or by accessing the locations pointed to by the dereferenced URIs of classes and properties in the data. We assume for brevity that all of the pertinent terminological data have already been collected and exist in the input data. If T-Box data are sourced separately via different means we can build an in-memory representation directly, without requiring the first scan of all input data.

We apply the following algorithm to create the T-Box in-memory representation, which we will analyse in the following sections:

1. FULL SCAN 1: separate T-Box information as described in Definition 24.

2. TBOX SCAN 1 & 2: reduce irrelevant RDF collection statements.

3. TBOX SCAN 3: perform authoritative analysis of the T-Box data and load in-memory representation.

4.3.1 Separating and Reducing T-Box Data

Firstly, we wish to separate all possible T-Box statements from the main bulk of data. $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$ are stored in memory and then the data dump is scanned. Quadruples with property $\in \mathcal{P}_{SAOR} \cup \{$`rdf:first`, `rdf:rest` $\}$ or `rdf:type` statements with object $\in \mathcal{C}_{SAOR}$ (which, where applicable, abide by the OWL abstract syntax) are buffered to a T-Box data file.

However, the T-Box data file still contains a large amount of RDF collection statements (property $\in \{$ `rdf:first, rdf:rest` $\}$) which are not related to reasoning. SAOR is only interested in such statements wherein they form part of a `:unionOf`, `:intersectionOf` or `:oneOf` class description. Later when the T-Box is being loaded, these collection fragments are reconstructed in-memory and irrelevant collection fragments are discarded; to reduce the amount of memory required we can quickly discard irrelevant collection statements through two T-Box scans:

- scan the T-Box data and store contexts of statements where the property $\in$ { `:unionOf`, `:intersectionOf`, `:oneOf` }.
- scan the T-Box data again and remove statements for which both hold:
  - property $\in$ { `rdf:first, rdf:rest` }
  - the context does not appear in those stored from the previous scan.

These scans quickly remove irrelevant collection fragments where a `:unionOf`, `:intersectionOf`, `:oneOf` statement does not appear in the same source as the fragment (i.e., collections which cannot contribute to the T-Box pattern of one of our rules).

4.3.2 Authoritative Analysis

We next apply authoritative analysis to the T-Box and load the results into our in-memory representation; in other words, we build an *authoritative T-Box* which pre-computes authority of T-Box data. We denote our authoritative T-Box by $\widehat{\mathbb{T}}$, whereby $Cl_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB}) = Cl_{\mathcal{R}}(\widehat{\mathbb{T}}, \mathbb{KB})$; for each rule, $\widehat{\mathbb{T}}$ only contains T-Box pattern instances for $Ante_{\mathcal{T}}$ which can lead to an authoritative rule application.

Each statement read is initially matched without authoritative analysis against the patterns enumerated in Table 3. If a pattern is initially matched, the positions required to be authoritative, as identified in boldface, are checked. If one such authoritative check is satified, the pattern is loaded into the T-Box. Indeed the same statement may be matched by more than one T-Box pattern for different rules with different authoritative restrictions; for example the statement `foaf:name :equivalentProperty` **`my:name`** `.` retrieved from `my:` namespace matches the T-Box pattern of rules **rdfp13a′** & **rdfp13b′**, but only conforms to the authoritative restriction for rule **rdfp13b′**. Therefore, we only store the statement in such a fashion as to apply to rule **rdfp13b′**; that is, the authoritative T-Box stores T-Box pattern instances separately for each rule, according to the authoritative restrictions for that rule.

Checking the authority of a source for a given namespace URI, as presented in Definition 29, may require a HTTP connection to the namespace URI so as to determine whether a redirect exists to the authoritative document (HTTP Response Code 303). Results of accessing URIs are cached once in-memory so as to avoid establishing repetitive connections. If the pattern is authoritatively matched, the statement is reflected in the in-memory T-Box. Alternatively, where available, a crawler can provide a set of redirect pairs which can be loaded into the system to avoid duplicating HTTP lookups; we presume for generality that such information is not provided.

4.3.3 In-Memory T-Box

Before we proceed, we quickly discuss the storage of `:oneOf` constructs in the T-Box for rule **rdfc0**. Individuals `(?x`$_1$ `...` `?x`$_n$`)` are stored with pointers to the one-of class **`?c`**. Before input data are read, these individuals are asserted to be of the `rdf:type` of their encompassing one-of class.

Besides the one-of support, for the in-memory T-Box we employ two separate hashtables, one for classes and another for properties, with RDF terms as key and a Java representation of the class or property as value. The representative Java objects contain labelled links to related objects as defined in Table 3. The property and

	$\mathcal{R}0$	
rdfc0	?C :oneOf (?x$_1$ ... ?x$_n$) .	(?x$_1$ ... ?x$_n$) $\xrightarrow{\mathbf{rdfc0}}$?C
	$\mathcal{R}1$	
rdfs2	**?P** rdfs:domain ?C .	**?P** $\xrightarrow{\mathbf{rdfs2}}$?C
rdfs3′	**?P** rdfs:range ?C .	**?P** $\xrightarrow{\mathbf{rdfs3'}}$?C
rdfs7′	**?P** rdfs:subPropertyOf ?Q .	**?P** $\xrightarrow{\mathbf{rdfs7'}}$?Q
rdfs9	**?C** rdfs:subClassOf ?D .	**?C** $\xrightarrow{\mathbf{rdfs9}}$?D
rdfp3′	**?P** a :SymmetricProperty .	**?P** $\xrightarrow{\mathbf{rdfp3'}}$ TRUE
rdfp8a′	**?P** :inverseOf ?Q .	**?P** $\xrightarrow{\mathbf{rdfp8a'}}$?Q
rdfp8b′	?P :inverseOf **?Q** .	**?Q** $\xrightarrow{\mathbf{rdfp8b'}}$?P
rdfp12a′	**?C** :equivalentClass ?D .	**?C** $\xrightarrow{\mathbf{rdfp12a'}}$?D
rdfp12b′	?C :equivalentClass **?D** .	**?D** $\xrightarrow{\mathbf{rdfp12b'}}$?C
rdfp13a′	**?P** :equivalentProperty ?Q .	**?P** $\xrightarrow{\mathbf{rdfp13a'}}$?Q
rdfs13b′	?P :equivalentProperty **?Q** .	**?Q** $\xrightarrow{\mathbf{rdfs13b'}}$?P
rdfp14a′	?C :hasValue ?y ; :onProperty **?P** .	**?P** $\xrightarrow{\mathbf{rdfp14a'}}$ { ?C, ?y }
rdfp14b′	?C :hasValue ?y ; :onProperty ?P .	?C $\xrightarrow{\mathbf{rdfp14b'}}$ { ?P, ?y }
rdfc1	?C :unionOf (?C$_1$...**?C$_i$**...?C$_n$) .	**?C$_i$** $\xrightarrow{\mathbf{rdfc1}}$?C
rdfc2	?C :minCardinality 1 ; :onProperty **?P** .	**?P** $\xrightarrow{\mathbf{rdfc2}}$?C
rdfc3a	?C :intersectionOf (?C$_1$ ... ?C$_n$) .	?C $\xrightarrow{\mathbf{rdfc3a}}$ { ?C$_1$, ..., ?C$_n$ }
rdfc3b	?C :intersectionOf (**?C$_1$**) .	**?C$_1$** $\xrightarrow{\mathbf{rdfc3b}}$?C
	$\mathcal{R}2$	
rdfp1′	**?P** a :FunctionalProperty .	**?P** $\xrightarrow{\mathbf{rdfp1'}}$ TRUE
rdfp2	**?P** a :InverseFunctionalProperty .	**?P** $\xrightarrow{\mathbf{rdfp2}}$ TRUE
rdfp4	**?P** a :TransitiveProperty .	**?P** $\xrightarrow{\mathbf{rdfp4}}$ TRUE
rdfp15′	?C :someValuesFrom **?D** ; :onProperty **?P** .	**?P** $\overset{\mathbf{rdfp15'}}{\leftrightarrow}$ **?D** $\xrightarrow{\mathbf{rdfp15'}}$?C
rdfp16′	?C :allValuesFrom ?D ; :onProperty ?P .	?P $\overset{\mathbf{rdfp16'}}{\leftrightarrow}$?C $\xrightarrow{\mathbf{rdfp16'}}$?D
rdfc3c	?C :intersectionOf (**?C$_1$ ... ?C$_n$**) .	{ **?C$_1$, ..., ?C$_n$** } $\xrightarrow{\mathbf{rdfc3c}}$?C
rdfc4a	?C :cardinality 1 ; :onProperty ?P .	?C $\overset{\mathbf{rdfc4a}}{\leftrightarrow}$?P
rdfc4b	?C :maxCardinality 1 ; :onProperty ?P .	?C $\overset{\mathbf{rdfc4b}}{\leftrightarrow}$?P

Table 3: T-Box statements and how they are used to *wire* the concepts contained in the in-memory T-Box.

class objects are designed to contain all of the information required for reasoning on a membership assertion of that property or class: that is, classes/properties satisfying the A-Box antecedent pattern of a rule are linked to the classes/properties appearing in the consequent of that rule, with the link labelled according to that rule. During reasoning, the class/property identifier used in the membership assertion is sent to the corresponding hashtable and the returned internal object used for reasoning on that assertion. The objects contain the following:

- Property objects contain the property URI and references to objects representing domain classes (**rdfs2**), range classes (**rdfs3′**), super properties (**rdfs7′**), inverse properties (**rdfs8***) and equivalent properties (**rdfp13***). References are kept to restrictions where the property in question is the object of an `:onProperty` statement (**rdfp14a**, **rdfp16′**, **rdfc2**, **rdfc4***). Where applicable, if the property is part of a some-values-from restriction, a pointer is kept to the some-values-from class (**rdfp15′**). Boolean values are stored to indicate whether the property is functional (**rdfp1′**), inverse-functional (**rdfp2**), symmetric (**rdfp3′**)

and/or transitive (**rdfp4**).

- Class objects contain the class URI and references to objects representing super classes (**rdfs9**), equivalent classes (**rdfp12***) and classes for which this class is a component of a union (**rdfc1**) or intersection (**rdfc3b/c**). On top of these core elements, different references are maintained for different types of class description:
  - intersection classes store references to their constituent class objects (**rdfc3a**)
  - restriction classes store a reference to the property the restriction applies to (**rdfp14b′**, **rdfp15′**, **rdfc2**, **rdfc4***) and also, if applicable to the type of restriction:
    - the values which the restriction property must have (**rdfp14b′**)
    - the class for which this class is a some-values-from restriction value (**rdfp15′**)

Figure 2 provides a UML-like representation of our T-Box, including multiplicities of the various links present between classes, properties and individuals labelled according to Table 3 for each rule.

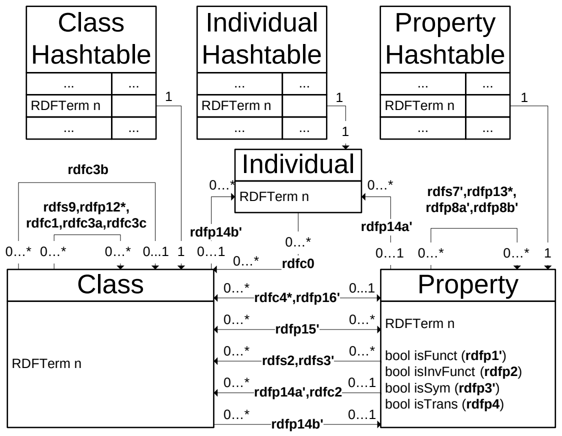


Figure 2: In-memory T-Box structure

The algorithm must also performs in-memory joining of collection segments according to `rdf:first` and `rdf:rest` statements found during the scan for the purposes of building union, intersection and enumeration class descriptions. Again, any remaining collections not relevant to the T-Box segment of the knowledge-base (i.e., not terminological collection statements) are discarded at the end of loading the input data; we also discard cyclic and branching lists as well as any lists not found to end with the `rdf:nil` construct.

We have now loaded the final T-Box for reasoning into memory; this T-Box will remain fixed throughout the whole reasoning process.

4.4 Initial Input Scan

Having loaded the terminological data, SAOR is now prepared for reasoning by statement-wise scan of the assertional data.

```
Input: statement s
Global: $\mathbb{T}$, index, output
foreach rule r $\in \mathcal{R}1$ do
    fire rule r w.r.t s and $\mathbb{T}$;
    foreach inferred statement: i do
        if i is unique then
            if i is valid RDF then
                write to output;
            ReasonStatement (i) ;
foreach rule r $\in \mathcal{R}2 \cup \mathcal{R}3$ do
    if s relevant for r w.r.t. $\mathbb{T}$ then
        write s to $\mathsf{index}_\mathsf{r}$ ;
```

Figure 3: ReasonStatement(s)

We provide the high-level flow for reasoning over an input statement s in Function ReasonStatement(s), cf. Figure 3. The reasoning scan process can be described as recursive depth-first reasoning whereby each unique statement produced is again input immediately for reasoning. Statements produced thus far for the original input statement are kept in a set to provide uniqueness testing and avoid cycles; a uniquing function is also maintained for a common subject group in the data, ensuring that statements are only produced once for that statement group. Once all of the statements produced by a rule have been themselves recursively analysed, the reasoner moves on to analysing the proceeding rule and loops until no unique statements are inferred. The reasoner then processes the next input statement.

There are three disjoint categories of statements which require different handling: namely (i) `rdf:type` statements, (ii) `:sameAs` statements, (iii) all other statements. We assume disjointness between the statement categories: we do not allow any external extension of the core `rdf:type`/`:sameAs` semantics (non-standard use / non-authoritative extension). Further, the assertions about `rdf:type` in the RDFS specification define the `rdfs:domain` and `rdfs:range` of `rdf:type` as being `rdfs:Resource` and `rdfs:Class`; since we are not interested in inferring membership of such RDFS classes we do not subject `rdf:type` statements to property-based entailments. The only assertions about `:sameAs` from the OWL specification define domain and range as `:Thing` which we ignore by the same justification.

The `rdf:type` statements are subject to class-based entailment reasoning and require joins with class descriptions in the T-Box. The `:sameAs` statements are handled by ruleset $\mathcal{R}3$, which we discuss in Section 4.6. All other statements are subject to property-based entailments and thus requires joins with T-Box property descriptions.

Ruleset $\mathcal{R}2 \cup \mathcal{R}3$ cannot be computed solely on a statement-wise basis. Instead, for each rule, we assign an on-disk file (blocked and compressed to save disk space). Each file contains statements which may contribute to satisfying the antecedent of its pertinent rule. During the scan, if an A-Box statement satisfies the necessary T-Box join for a rule, it is written to the index for that rule. For example, when the statement

```
ex:me foaf:isPrimaryTopicOf ex:myHomepage .
```

is processed, the property object for `foaf:isPrimaryTopicOf` is retrieved from the T-Box property hashtable. The object states that this property is of type `:InverseFunctionalProperty` ($\overset{\textbf{rdfp2}}{\rightarrow}$ TRUE). The rule cannot yet be fired as this statement alone does not satisfy the A-Box segment of the antecedent of **rdfp2** and the method is privy to only one A-Box statement at a time. When, later, the statement:

```
ex:me2 foaf:isPrimaryTopicOf ex:myHomepage .
```

is found, it also is written to the same file – the file now contains sufficient data to (although it cannot yet) fire the rule and infer:

```
ex:me :sameAs ex:me2 .
```

During the initial scan and inferencing, all files for ruleset $\mathcal{R}2 \cup \mathcal{R}3$ are filled with pertinent statements analogously to the example above. After the initial input statements have been exhausted, these files are analysed to infer, for example, the `:sameAs` statement above.

4.5 On-Disk A-Box Join Analysis

In this section, we discuss handling of the on-disk files containing A-Box statements for ruleset $\mathcal{R}2 \cup \mathcal{R}3$. We firstly give a general overview of the execution for each rule using an on-disk file and then look at the execution of each rule.

	$\mathcal{R}2$	
rdfp1′	**?x** ?P ?y , ?z .	SPOC
rdfp2	?x ?P **?z** . ?y ?P **?z** .	OPSC
rdfp4	?x ?P **?y** . **?y** ?P ?z .	SPOC & OPSC
rdfp15′	$\underline{\text{?x ?P }\textbf{?y}\text{ .}}$ **?y** a ?D .	SPOC / $\underline{\text{OPSC}}$
rdfp16′	**?x** a ?C ; ?P ?y .	SPOC
rdfc3c	**?x** a ?C$_1$, ..., ?C$_n$.	SPOC
rdfc4a	**?x** a ?C ; ?P ?y , ?z .	SPOC
rdfc4b	**?x** a ?C ; ?P ?y , ?z .	SPOC
	$\mathcal{R}3$	
rdfp7	?x :sameAs **?y** . **?y** :sameas ?z .	SPOC & OPSC
rdfp11′	**?x** :sameAs ?_x ; ?P ?y .	SPOC
rdfp11″	**?y** :sameAs ?_y . $\underline{\text{?x ?P }\textbf{?y}\text{ .}}$	SPOC / $\underline{\text{OPSC}}$

Table 4: Table enumerating the A-Box joins to be computed using the on-disk files with key join position in boldface font and sorting order required for statements to compute join.

Table 4 presents the joins to be executed via the on-disk files for each rule: the key join variables, used for computing the join, are shown in boldface. In this table we refer to *SPOC and OPSC sorting order*: these can be intuitively interpreted as quads sorted according to subject, predicate, object, context (natural sorting order) and object, predicate, subject, context (inverse sorting order) respectively. For the internal index files, we use context to encode the sorting order of a statement and the iteration in which it was added; only joins with at least one new statement from the last iteration will infer novel output.

Again, an on-disk file is dedicated for each rule/join required. The joins to be computed are a simple "star shaped" join pattern or "one-hop" join pattern (which we reduce to a simple star shaped join computation by inverting one one or more patterns to inverse order). The statements in each file are initially sorted according to the key join variable. Thus, common bindings for the key join variable are grouped together and joins can be executed by means of sequential scan for common key join variable binding groups.

We now continue with a more detailed description of the process for each rule beginning with the more straightforward rules.

4.5.1 Functional Property Reasoning - Rule rdfp1′

From the initial input scan, we have a file containing only statements with functional properties in the predicate position (as described in Section 4.4). As can be seen from Table 4, the key join variable is in the subject position for all A-Box statements in the pattern. Thus, we can sort the file according to SPOC (natural) order. The result is a file where all statements are grouped according to a common subject, then predicate, then object. We can now scan this file, storing objects with a common subject-predicate. We can then fire the rule stating equivalence between these objects.

4.5.2 Inverse Functional Reasoning - Rule rdfp2

Reasoning on statements containing inverse functional properties is conducted analogously to functional property reasoning. However, the key join variable is now in the object position for all A-Box statements in the pattern. Thus, we instead sort the file according to OPSC (inverse) order and scan the file inferring equivalence between the subjects for a common object-predicate group.

4.5.3 Intersection Class Reasoning - Rule rdfc3c

The key join variable for rule **rdfc3c** is in the subject position for all A-Box triple patterns. Thus we can sort the file for the rule (filled with memberships assertions for classes which are part of some intersection) according to SPOC order. We can scan common subject-predicate (in any case, the predicates all have value `rdf:type`) groups storing the objects (all types for the subject resource which are part of an intersection). The containing intersection for each type can then be retrieved (through $\overset{\textbf{rdfc3c}}{\rightarrow}$) and the intersection checked to see if all of it's constituent types have been satisfied. If so, membership of the intersection is inferred.

4.5.4 All-Values-From Reasoning - Rule rdfp16′

Again, the key join variable for rule **rdfp16′** is in the subject position for all A-Box triple patterns and again we can sort the file according to SPOC order. For a common subject group, we store `rdf:type` values and also all predicate/object edges for the given subject. For every member of an all-values-from restriction class (as is given by all of the `rdf:type` statements in the file according to the join with the T-Box on the

?C position), we wish to infer that objects of the :onProperty value (as is given by all the non-rdf:type statements according to the T-Box join with ?P – where ?P is linked from ?C with $\overset{\textbf{rdfp16}'}{\rightarrow}$) are of the all-values-from class. Therefore, for each restriction membership assertion, the objects of the corresponding :onProperty-value membership-assertions are inferred to be members of the all-values-from object class (?D).

4.5.5 Some-Values-From Reasoning - Rule rdfp15′

For some-values-from reasoning, the key join variable is in the subject position for rdf:type statements (all membership assertions of a some-values-from object class) but in the object position for the :onProperty value membership assertions. Thus, we order class membership assertions in the file according to natural SPOC order and property membership assertions according to inverse OPSC order. In doing so, we can scan common **?y** binding groups in the file, storing rdf:type values and also all predicate/subject edges. For every member of a some-values-from object class (as is given by all of the rdf:type statements in the file according to the join with the T-Box on the ?D position), we infer that subjects of the :onProperty-value statements (as is given by all the non-rdf:type statements according to the T-Box join with ?P) are members of the restriction class (?C).

4.5.6 Transitive Reasoning (Non-Symmetric) - Rule rdfp4

Transitive reasoning is perhaps the most challenging to compute: the output of rule **rdfp4** can again recursively act as input to the rule. For closure, recursive application of the rule must be conducted in order to traverse arbitrarily long transitive paths in the data.

Firstly, we will examine sorting order. The key join variable is in the subject position for one pattern and in the object position for the second pattern. However, both patterns are identical: a statement which matches one pattern will obviously match the second. Thus, every statement in the transitive reasoning file is duplicated with one version sorted in natural SPOC order, and another in inverse OPSC.

Take, for example, the following triples where ex:comesBefore is asserted in the T-Box as being of type :TransitiveProperty:

# INPUT:

```
ex:a ex:comesBefore ex:b .
ex:b ex:comesBefore ex:c .
ex:c ex:comesBefore ex:d .
```

In order to compute the join, we must write the statements in both orders, using the context to mark which triples are in inverse order, and sort them accordingly (for this internal index, we temporarily relax the requirement that context is a URI).

# SORTED FILE - ITERATION 1:[13]

```
ex:a ex:comesBefore ex:b _:spoc1 .
ex:b ex:comesBefore ex:a _:opsc1 .
ex:b ex:comesBefore ex:c _:spoc1 .
ex:c ex:comesBefore ex:b _:opsc1 .
```

[13]In N-Quads format: c.f. http://sw.deri.org/2008/07/n-quads/

```
ex:c ex:comesBefore ex:d _:spoc1 .
ex:d ex:comesBefore ex:c _:opsc1 .
```

The data, as above, can then be scanned and for each common join-binding/predicate group (e.g., `ex:b ex:comesBefore`), the subjects of statements in inverse order (e.g., `ex:a`) can be linked to the object of naturally ordered statements (e.g., `ex:c`) by the transitive property. However, such a scan will only compute a single one-hop join. From above, we only produce:

# OUTPUT - ITERATION 1 / INPUT - ITERATION 2

```
ex:a ex:comesBefore ex:c .
ex:b ex:comesBefore ex:d .
```

We still not have not computed the valid statement `ex:a ex:comesBefore ex:d .` which requires a two hop join. Thus we must iteratively feedback the results from one scan as input for the next scan. The output from the first iteration, as above, is also reordered and sorted as before and merge-sorted into the main ***SORTED FILE***.

# SORTED FILE - ITERATION 2:

```
ex:a ex:comesBefore ex:b _:spoc1 .
ex:a ex:comesBefore ex:c _:spoc2 .
ex:b ex:comesBefore ex:a _:opsc1 .
ex:b ex:comesBefore ex:c _:spoc1 .
ex:b ex:comesBefore ex:d _:spoc2 .
ex:c ex:comesBefore ex:a _:opsc2 .
ex:c ex:comesBefore ex:b _:opsc1 .
ex:c ex:comesBefore ex:d _:spoc1 .
ex:d ex:comesBefore ex:b _:opsc2 .
ex:d ex:comesBefore ex:c _:opsc1 .
```

The observant reader may already have noticed from above that we also mark the context with the iteration for which the statement was added. In every iteration, we only compute inferences which involve the delta from the last iteration; thus the process is comparable to semi-naïve evaluation. Only joins containing at least one newly added statement are used to infer new statements for output. Thus, from above, we avoid repeat inferences from ***ITERATION 1*** and instead infer:

# OUTPUT - ITERATION 2:

```
ex:a ex:comesBefore ex:d .
```

A fixpoint is reached when no new statements are inferred. Thus we would require another iteration for the above example to ensure that no new statements are inferable. The number of iterations required is in $\mathcal{O}(\log n)$ according to the longest unclosed transitive path in the input data. Since the algorithm requires scanning of not only the delta, but also the entire data, performance using on-disk file scans alone would be sub-optimal. For example, if one considers that most of the statements constitute paths of, say $\leq$8 vertices, one path containing 128 vertices would require four more scans after the bulk of the paths have been closed.

With this in mind, we accelerate transitive closure by means of an in-memory transitivity index. For each transitive property found, we store sets of linked lists which represent the graph extracted for that property. From the example ***INPUT*** from above, we would store.

```
ex:comesBefore -- ex:a -> ex:b -> ex:c -> ex:d
```

From this in-memory linked list, we would then collapse all paths of length ≥ 2 (all paths of length 1 are input statements) and infer closure at once:

# OUTPUT - ITERATION 1 / INPUT - ITERATION 2

```
ex:a ex:comesBefore ex:c .
ex:a ex:comesBefore ex:d .
ex:b ex:comesBefore ex:d .
```

Obviously, for scalability requirements, we do not expect the entire transitive body of statements to fit in-memory. Thus, before each iteration we calculate the in-memory capacity and only store a pre-determined number of properties and vertices. Once the in-memory transitive index is full, we infer the appropriate statements and continue by file-scan. The in-memory index is only used to store the delta for a given iteration (everything for the first iteration). Thus, we avoid excess iterations to compute closure of a small percentage of statements which form a long chain and greatly accelerate the fixpoint calculation.

4.5.7 Transitive Reasoning (Symmetric) - Rules rdfp3′/rdfp4

We use a separate on-disk file for membership assertions of properties which are both transitive *and* symmetric. A graph of symmetric properties is direction-less, thus the notion of direction as evident above though use of inverted ordered statements is unnecessary. Instead, all statements and their inverses (computed from symmetric rule **rdfp3′**) are written in natural SPOC order and direct paths are inferred between all objects in a common subject/predicate group. The in-memory index is again similar to above; however, we instead use a direction-less doubly-linked list.

4.6 Equality Reasoning

Thus far, we have not considered `:sameAs` entailment, which is supported in SAOR through rules in $\mathcal{R}3$. Prior to executing rules **rdfp11′** & **rdfp11″**, we must first perform symmetric transitive closure on the list of all `:sameAs` statements (rules **rdfp6′** & **rdfp7**). Thus, we use an on-disk file analogous to that described in Section 4.5.7.

However, for rules **rdfp6′** & **rdfp7**, we do not wish to experience an explosion of inferencing through long equivalence chains (lists of equivalent individuals where there exists a `:sameAs` path from each individual to every other individual). The closure of a symmetric transitive chain of n vertices results in $n(n-1)$ edges or statements (ignoring reflexive statements). For example, in [23] we found a chain of 85,803 equivalent individuals inferable from a web dataset.[14] Naïvely applying symmetric transitive reasoning as discussed in Section 4.5.7 would result in a closure of 7.362b `:sameAs` statements for this chain alone.

Similarly, `:sameAs` entailment, as according to rules **rdfp11′** & **rdfp11″**, duplicates data for all equivalent individuals which could result in a massive amount of duplicate data (particularly when considering uniqueness on a quad level: i.e., including duplicate triples from different sources). For example, if each of the 85,803 equivalent individuals had attached an average of 8 unique statements, then this could equate to 8*85,803*85,803 = 59b inferred statements.

[14]This is from incorrect use of the FOAF ontology by prominent exporters. We refer the interested reader to [23]

Obviously, we must avoid the above scenarios, so we break from complete inference with respect to the rules in $\mathcal{R}3$. Instead, for each set of equivalent individuals, we chose a pivot identifier to use in rewriting the data. The pivot identifier is used to keep a consistent identifier for the set of equivalent individuals: the alphabetically highest pivot is chosen for convenience of computation. For alternative choices of pivot identifiers on web data see [23]. We use the pivot identifier to consolidate data by rewriting all occurrences of equivalent identifiers to the pivot identifier (effectively merging the equivalent set into one individual).

Thus, we do not derive the entire closure of `:sameAs` statements as indicated in rules **rdfp6′** & **rdfp7** but instead only derive an equivalence list which points from equivalent identifiers to their pivots. As highlighted, use of a pivot identifier is necessary to reduce the amount of output statements, effectively compressing equivalent resource descriptions: we hint here that a fully expanded view of the descriptions could instead be supported through backward-chaining over the semi-materialised data.

To achieve the pivot compressed inferences we use an on-disk file containing `:sameAs` statements. Take for example the following statements:

# INPUT

```
ex:a :sameAs ex:b .
ex:b :sameAs ex:c .
ex:c :sameAs ex:d .
```

We only wish to infer the following output for the pivot identifier `ex:a`:

# OUTPUT PIVOT EQUIVALENCES

```
ex:b :sameAs ex:a .
ex:c :sameAs ex:a .
ex:d :sameAs ex:a .
```

The process is the same as that for symmetric transitive reasoning as described before: however, we only close transitive paths to nodes with the highest alphabetical order. So, for example, if we have already materialised a path from `ex:d` to `ex:a` we ignore inferring a path from `ex:d` to `ex:b` as `ex:b` $>$ `ex:a`.

To execute rules **rdfp11′** & **rdfp11″** and perform "consolidation" (rewriting of equivalent identifiers to their pivotal form), we perform a zig-zag join: we sequentially scan the `:sameAs` inference output as above and an appropriately sorted file of data, rewriting the latter data according to the `:sameAs` statements. For example, take the following statements to be consolidated:

# UNCONSOLIDATED DATA

```
ex:a foaf:mbox <mail@example.org> .
...
ex:b foaf:mbox <mail@example.org> .
ex:b foaf:name "Joe Bloggs" .
...
ex:d :sameAs ex:b .
...
ex:e foaf:knows ex:d .
```

The above statements are scanned sequentially with the closed `:sameAs` pivot output from above. For example, when the statement `ex:b foaf:mbox <mailto:mail-@example.org> .` is first read from the unconsolidated data, the `:sameAs` index is scanned until `ex:b :sameAs ex:a .` is found (if `ex:b` is not found in the `:sameAs` file, the scan is paused when an element above the sorting order of `ex:b` is found). Then, `ex:b` is rewritten to `ex:a`.

# PARTIALLY CONSOLIDATED DATA

```
ex:a foaf:mbox <mail@example.org> .
...
ex:a foaf:mbox <mail@example.org> .
ex:a foaf:name "Joe Bloggs" .
...
ex:a :sameAs ex:b .
...
ex:e foaf:knows ex:d .
```

We have now executed rule **rdfp11′** and have the data partially consolidated as shown. However, the observant reader will notice that we have not consolidated the object of the last two statements. We must sort the data again according to inverse OPSC order and again sequentially scan both the partially consolidated data and the `:sameAs` pivot equivalences, this time rewriting `ex:b` and `ex:d` in the object position to `ex:a` and producing the final consolidated data. This equates to executing rule **rdfp11″**.

For the purposes of the on-disk files for computing rules requiring A-Box joins, we must consolidate the key join variable bindings according to the `:sameAs` statements found during reasoning. For example consider the following statements in the functional reasoning file:

```
ex:a ex:mother ex:m1 .
ex:b ex:mother ex:m2 .
```

Evidently, rewriting the key join position according to our example pivot file will lead to inference of:

```
ex:m1 :sameAs ex:m2 .
```

which we would otherwise miss. Thus, whenever the index of `:sameAs` statements is changed, for the purposes of closure it is necessary to attempt to rewrite all join index files according to the new `:sameAs` statements. Since we are, for the moment, only concerned with consolidating on the join position we need only apply one consolidation scan.

The final step in the SAOR reasoning process is to finalise consolidation of the initial input data and the newly inferred output statements produced by all rules from scanning and on-disk file analysis. Although we have provided exhaustive application of all inferencing rules, and we have the complete set of `:sameAs` statements, elements in the input and output files may not be in their equivalent pivotal form. Therefore, in order to ensure proper consolidation of all of the data according to the final set of `:sameAs` statements, we must firstly sort both input and inferred sets of data in SPOC order, consolidate subjects according to the pivot file as above; sort according to OPSC order and consolidate objects.

However, one may notice that `:sameAs` statements in the data become consolidated into reflexive statements: i.e., from the above example `ex:a :sameAs ex:a`. Thus, for the final output, we remove any `:sameAs` statements in the data and instead merge the statements contained in our final pivot `:sameAs` equivalence index, and their inverses, with the consolidated data. These statements retain the list of all possible identifiers for a consolidated entity in the final output.

4.7 Achieving Closure

We conclude this section by summarising the approach, detailing the overall fixpoint calculations (as such, putting the jigsaw together) and detailing how closure is achieved using the individual components. Along these lines, in Figure 4 we provide a summary of the algorithmic steps seen so far and, in particular, show the fixpoint calculations involved for exhaustive application of ruleset $\mathcal{R}2 \cup \mathcal{R}3$; we compute one main fixpoint over all of the operations required, within which we also compute two local fixpoints.

Firstly, since all rules in $\mathcal{R}2$ are dependant on `:sameAs` equality, we perform `:sameAs` inferences first. Thus, we begin closure on $\mathcal{R}2 \cup \mathcal{R}3$ with a local equality fixpoint which (i) executes all rules which produce `:sameAs` inferences (**rdfp1′**,**rdfp2**,**rdfc4***); (ii) performs symmetric-transitive closure using pivots on all `:sameAs` inferences; (iii) rewrites **rdfp1′**, **rdfp2** and **rdfc4*** indexes according to `:sameAs` pivot equivalences and (iv) repeats until no new `:sameAs` statements are produced.

Next, we have a local transitive fixpoint for recursively computing transitive property reasoning: (i) the transitive index is rewritten according to the equivalences found through the above local fixpoint; (ii) a transitive closure iteration is run, output inferences are recursively fed back as input; (iii) ruleset $\mathcal{R}1$ is also recursively applied over output from previous step whereby the output from ruleset $\mathcal{R}1$ may also write new statements to any $\mathcal{R}2$ index. The local fixpoint is reached when no new transitive inferences are computed.

Finally, we conclude the main fixpoint by running the remaining rules: **rdfp15′**, **rdfp16′** and **rdfc3c**. For each rule, we rewrite the corresponding index according to the equivalences found from the first local fixpoint, run the inferencing over the index and send output for reasoning through ruleset $\mathcal{R}1$. Statements inferred directly from the rule index, or through subsequent application of ruleset $\mathcal{R}1$, may write new statements for $\mathcal{R}2$ indexes. This concludes one iteration of the main fixpoint, which is run until no new statements are inferred.

For each ruleset $\mathcal{R}0 - 3$, we now justify our algorithm in terms of our definition of closure with respect to our static T-Box. Firstly, closure is achieved immediately upon ruleset $\mathcal{R}0$, which requires only T-Box knowledge, from our static T-Box. Secondly, with respect to the given T-Box, every input statement is subject to reasoning according to ruleset $\mathcal{R}1$, as is every statement inferred from ruleset $\mathcal{R}0$, those recursively inferred from ruleset $\mathcal{R}1$ itself, and those recursively inferred from on-disk analysis for ruleset $\mathcal{R}1 \cup \mathcal{R}2$. Next, every input statement is subject to reasoning according to ruleset $\mathcal{R}2$ with respect to our T-Box; these again include all inferences from $\mathcal{R}0$, all statements inferred through $\mathcal{R}1$ alone, and all inferences from recursive application of ruleset $\mathcal{R}1 \cup \mathcal{R}2$.

Therefore, we can see that our algorithm applies exhaustive application of ruleset $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$ with respect to our T-Box, leaving only consideration of equality reasoning in ruleset $\mathcal{R}3$. Indeed, our algorithm is not complete with respect to ruleset $\mathcal{R}3$ since we choose pivot identifiers for representing equivalent individuals as justified in Section 4.6. However, we still provide a form of "pivotal closure" whereby backward-chaining support of rules **rdfp11′** and **rdfp11″** over the output of our algorithm would provide a view of closure as defined; i.e., our output contains all of the possible inferences according to our notion of closure, but with equivalent individuals compressed in pivotal form.

Firstly, for rules **rdfp6′** and **rdfp7**, all statements where $p =$ `:sameAs` from the

```
Input: $\mathbb{KB}$
Output: $Cl_{\widehat{\mathcal{R}}}(\mathbb{T},\mathbb{KB})$
for scan $\mathbb{KB}$(Section 4.3.1) do
    obtain candidate statements for $\mathbb{T}$;
reduce $\mathbb{T}$, derive $\widehat{\mathbb{T}}$ (Section 4.3.2) ;
load $\widehat{\mathbb{T}}$ in-memory (Section 4.3.3) ;
run $\mathcal{R}0$ rules ;
for inferred statement i do
    if i is valid RDF then
        write to output;
    ReasonStatement (i) (Function 1) ;
for s $\in$ $\mathbb{KB}$(Section 4.4) do
    ReasonStatement (s) (Function 1) ;
output now contains all inferences on $\mathcal{R}0$ and $\mathcal{R}1$ for initial input ;
index now contain all initial information relevant for subsequent inferences on $\mathcal{R}2$ ;
sameas contains all initial information relevant for inferences on $\mathcal{R}3$ ;
repeat
    repeat
        for rule r $\in$ {rdfp1$'$,rdfp2,rdfc4*} (Section 4.5) do
            if new $_{r}$ is set then
                rewrite index $_{r}$ w.r.t. sameas/ unset new $_{r}$;
            if index $_{r}$ has changed or was rewritten then
                run r on index $_{r}$;
                write to sameas;
        if sameas has changed then
            run rules rdfp6$'$ and rdfp7 on sameas (Section 4.6) ;
            write sameas/ set all new;
    until fixpoint reached (no changes in previous iteration) ;
    if new $_{\mathbf{rdfp4}}$ is set then
        rewrite index $_{\mathbf{rdfp4}}$ w.r.t. sameas/ unset new $_{\mathbf{rdfp4}}$;
    repeat
        run rule rdfp4 on index $_{\mathbf{rdfp4}}$ (Section 4.5.6 and 4.5.7) ;
        for inferred statement i do
            write to index $_{\mathbf{rdfp4}}$ ;
            if i is RDF then
                write to output;
            ReasonStatement (i) (Function 1) ;
    until fixpoint reached (no changes in previous iteration) ;
    for rule $r \in$ {rdfp15$'$,rdfp16$'$,rdfc3c} (Section 4.5) do
        if new $_{r}$ is set then
            rewrite index $_{r}$ w.r.t. sameas/ unset new $_{r}$;
        if index $_{r}$ has changed or was rewritten then
            run r on index $_{r}$;
            for inferred statement i do
                if i is RDF then
                    write to output;
                ReasonStatement (i) (Function 1) ;
until fixpoint reached (no changes in previous iteration) ;
output contains inferred statements possibly in non-pivotal form ;
for subject and object (Section 4.6) do
    for scan $\mathbb{KB}$and output do
        rewrite according to sameas;
        write to $Cl_{\widehat{\mathcal{R}}}(\mathbb{T},\mathbb{KB})$;
    write sameas and sameas $^{-}$ to $Cl_{\widehat{\mathcal{R}}}(\mathbb{T},\mathbb{KB})$;
```

Figure 4: SAOR reasoning algorithm

original input or as produced by $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$ undergo on-disk symmetric-transitive closure in pivotal form. Since both rules only produce more `:sameAs` statements, and according to the standard usage restriction of our closure, they are not applicable to reasoning under $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$. Secondly, we loosely apply rules **rdfp11′** and **rdfp11″** such as to provide closure with respect to joins in ruleset $\mathcal{R}2$; i.e., all possible joins are computed with respect to the given `:sameAs` statements. Equivalence is clearly not important to $\mathcal{R}0$ since we strictly do not allow `:sameAs` statements to affect our T-Box; $\mathcal{R}1$ inferences do not require joins and, although the statements produced will not be in pivotal form, they will be output and rewritten later; inferences from $\mathcal{R}2$ will be produced as discussed, also possibly in non-pivotal form. In the final consolidation step, we then rewrite all statements to their pivotal form and provide incoming and outgoing `:sameAs` relations between pivot identifiers and their non-pivot equivalent identifiers. This constitutes our output, which we call *pivotal authoritative closure*.

5 Evaluation and Discussion

We now provide evaluation of the SAOR methodology firstly with quantitative analysis of the importance of authoritative reasoning, and secondly we provide performance measurements and discussion along with insights into the fecundity of each rule w.r.t. reasoning over web data. All experiments are run on one machine with a single Opteron 2.2 GHz CPU and 4 GB of main memory. We provide evaluation on two datasets: we provide complete evaluation for a dataset of 147m statements collected from 665k sources and scale-up experiments running scan-reasoning (rules in $\mathcal{R}0 \cup \mathcal{R}1$) on a dataset of 1.1b statements collected from 6.5m sources; both datasets are from web-crawls using MultiCrawler [21].

We create a unique set of blank nodes for each graph $\mathcal{W}' \in M(\mathcal{S}_w)$ using a function on c and the original blank node label which ensures a one-to-one mapping from the original blank node labels and uniqueness of the blank nodes for a given context c.

To show the effects of ontology hijacking we constructed two T-Boxes with and without authoritative analysis for each dataset. We then ran reasoning on single membership assertions for the top five classes and properties found natively in each dataset. Table 5 summarises the results. Taking `foaf:Person` as an example, with an authoritative T-Box, six statements are output for every input `rdf:type foaf:Person` statement in both datasets. With the non-authoritative T-Box, 388 and 4,631 statements are output for every such input statement for the smaller and larger datasets respectively. Considering that there are 3.25m and 63.33m such statements in the respective datasets, overall output for `rdf:type foaf:Person` input statements alone approach 1.26b and 293b statements for non-authoritative reasoning respectively. With authoritative reasoning we only produce 19.5m and 379.6m statements, a respective saving of 65x and 772x on output statement size.[15]

It should be noted that reasoning on a membership assertion of the top level class (`:Thing`/`rdfs:Resource`) is very large for both the 147m (234 inferences) and the 1.1b dataset (4251 inferences). For example, in both datasets, there are many `:unionOf`

[15]For example, the document retrievable from `http://pike.kw.nl/files/documents/pietzwart/RDF/PietZwart200602.owl` defines super-classes/-properties for all of the FOAF vocabulary.

147m Dataset

C	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$
`rss:item`	0	356	3,558,055	0	1,266,667,580
`foaf:Person`	6	388	3,252,404	19,514,424	1,261,932,752
`rdf:Seq`	2	243	1,934,852	3,869,704	470,169,036
`foaf:Document`	1	354	1,750,365	1,750,365	619,629,210
`wordnet:Person`	0	236	1,475,378	0	348,189,208
TOTAL	9	1,577	11,971,054	25,134,493	3,966,587,786

P	$\lvert Cl_{\mathcal{R}1}(\{\widehat{\mathbb{T}}, m(P)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\{\widehat{\mathbb{T}}, m(P)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$
`dc:title*`	0	14	5,503,170	0	77,044,380
`dc:date*`	0	377	5,172,458	0	1,950,016,666
`foaf:name*`	3	418	4,631,614	13,894,842	1,936,014,652
`foaf:nick*`	0	390	4,416,760	0	1,722,536,400
`rss:link*`	1	377	4,073,739	4,073,739	1,535,799,603
TOTAL	4	1,576	23,797,741	17,968,581	7,221,411,701

1.1b Dataset

C	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$
`foaf:Person`	6	4,631	63,271,689	379,630,134	293,011,191,759
`foaf:Document`	1	4,523	6,092,322	6,092,322	27,555,572,406
`rss:item`	0	4,528	5,745,216	0	26,014,338,048
`oboInOwl:DbXref`	0	0	2,911,976	0	0
`rdf:Seq`	2	4,285	2,781,994	5,563,988	11,920,844,290
TOTAL	9	17,967	80,803,197	391,286,444	358,501,946,503

P	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$
`rdfs:seeAlso`	2	8,647	113,760,738	227,521,476	983,689,101,486
`foaf:knows`	14	9,269	77,335,237	1,082,693,318	716,820,311,753
`dc:title*`	0	4,621	71,321,437	0	329,576,360,377
`foaf:nick*`	0	4,635	65,855,264	0	305,239,148,640
`foaf:weblog`	7	9,286	55,079,875	385,559,125	511,471,719,250
TOTAL	23	36,458	383,352,551	1,695,773,919	2,846,796,641,506

Table 5: Comparison of authoritative and non-authoritative reasoning for the number of unique inferred RDF statements produced (w.r.t. ruleset $\mathcal{R}1$) over the five most frequently occurring classes and properties in both input datasets. '*' indicates a datatype property where the object of $m(P)$ is a literal. The amount of statements produced for authoritative reasoning for a single membership assertion of the class or property is denoted by $\left|Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\right|$ and $\left|Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\right|$ respectively. Non-authoritative counts are given by $|Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})|$ and $|Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})|$. n is the number of membership assertions for the class C or property P in the given dataset.

class descriptions with `:Thing` as a member;[16] for the 1.1b dataset, many inferences on the top level classes stem from, for example, the OWL W3C Test Repository[17]. Of course we do not see such documents as being malicious in any way, but clearly they would cause inflationary inferences when naïvely considered as part of web knowledge-base.

Next, we present some metrics regarding the first step of reasoning: the separation and in-memory construction of the T-Box. For the 1.1b dataset, the initial scan of all data found 9,683,009 T-Box statements (0.9%). Reducing the T-Box by removing collection statements as described in Section 4.3.1 dropped a further 1,091,698 (11% of total) collection statements leaving 733,734 such statements in the T-Box (67% collection statements dropped) and 8,591,311 (89%) total. Table 6 shows, for membership assertions of each class and property in $\mathcal{C}_{SAOR}$ and $\mathcal{P}_{SAOR}$, the result of applying authoritative analysis. Of the 33,157 unique namespaces probed, 769 (2.3%) had a redirect, 4068 (12.3%) connected but had no redirect and 28,320 (85.4%) did not connect at all. In total, 14,227,116 authority checks were performed. Of these, 6,690,704 (47%) were negative and 7,536,412 (53%) were positive. Of the positive, 4,236,393 (56%) were blank-nodes, 2,327,945 (31%) were a direct match between namespace and source and 972,074 (13%) had a redirect from the namespace to the source. In total, 2,585,708 (30%) statements were dropped as they could not contribute to a valid authoritative inference. The entire process of separating, analysing and loading the T-Box into memory took 6.47 hours: the most costly operation here is the large amount of HTTP lookups required for authoritative analysis, with many connections unsuccessful after our five second timeout. The process required $\sim$3.5G of Java heap-space and $\sim$10M of stack space.

For the 147m dataset, 2,649,532 (1.7%) T-Box statements were separated from the data, which was reduced to 1,609,958 (61%) after reducing the amount of irrelevant collection statements; a further 536,564 (33%) statements were dropped as they could not contribute to a valid authoritative inference leaving 1,073,394 T-Box statements (41% of original). Loading the T-Box into memory took approximately 1.04 hours.

We proceed by evaluating the application of reasoning over all rules on the 147m dataset with respect to throughtput of statements written and read.

Figure 5 shows performance for reaching an overall fixpoint for application of all rules. Clearly, the performance plateaus after 79 mins. At this point the input statements have been exhausted, with rules in $\mathcal{R}0$ and $\mathcal{R}1$ having been applied to the input data and statements written to the on-disk files for $\mathcal{R}2$ and $\mathcal{R}3$. SAOR now switches over to calculating a fixpoint over the on-disk computed $\mathcal{R}2$ and $\mathcal{R}3$ rules, the results of which become the new input for $\mathcal{R}1$ and further recursive input to the $\mathcal{R}2$ and $\mathcal{R}3$ files.

Figure 6 shows performance specifically for achieving closure on the on-disk $\mathcal{R}2$ and $\mathcal{R}3$ rules. There are three pronounced steps in the output of statements. The first one shown at (a) is due to inferencing of `:sameAs` statements from rule **rdfp2** (`:InverseFunctionalProperty` - 2.1m inferences). Also part of the first step are `:sameAs` inferences from rules **rdfp1′** (`:FunctionalProperty` - 31k inferences) and rules **rdfc4*** (`:cardinality`/`:maxCardinality` - 449 inferences). For the first

[16]Fifty-five such `:unionOf` class descriptions can be found in `http://lsdis.cs.uga.edu/~oldham/ontology/wsag/wsag.owl`; 34 are in `http://colab.cim3.net/file/work/SICoP/ontac/reference/ProtegeOntologies/COSMO-Versions/TopLevel06.owl`.

[17]`http://www.w3.org/2002/03owlt/`

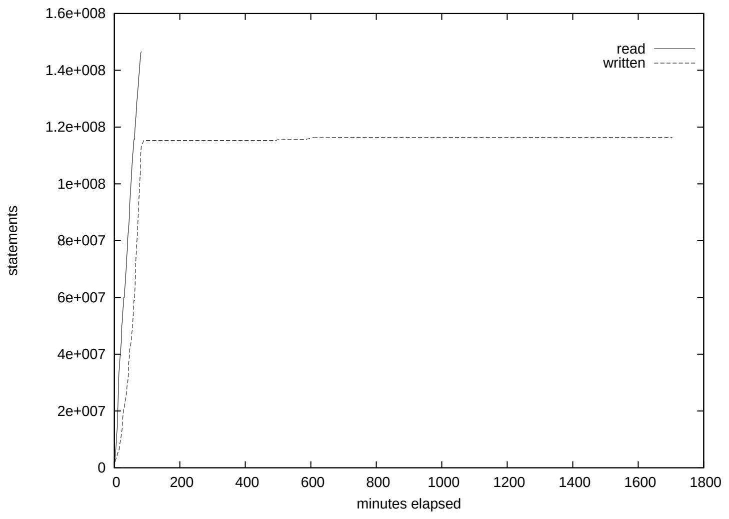


Figure 5: Performance of applying entire ruleset on the 147m statements dataset (without final consolidation step)

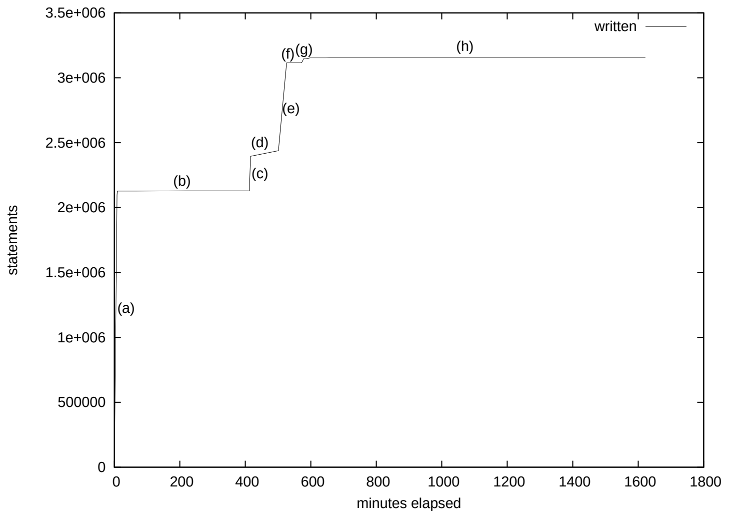


Figure 6: Performance of inferencing over $\mathcal{R}2$ and $\mathcal{R}3$ on-disk indexes for the 147m statements dataset (without final consolidation)

Property	AuthSub	AuthObj	AuthBoth	AuthNone	Total	Drop
`rdfs:subClassOf`	25,076	**583,399**	1,595,850	**1,762,414**	3,966,739	2,345,813
`:onProperty`	1,041,873	-	97,921	-	1,139,843	-
`:someValuesFrom`	681,968	-	217,478	-	899,446	-
`rdf:first`	273,805	-	392,707	-	666,512	-
`rdf:rest`	249,541	-	416,946	-	666,487	-
`:equivalentClass`	574	189,912	162,886	**3,198**	356,570	3,198
`:intersectionOf`	-	-	216,035	-	216,035	-
`rdfs:domain`	5,693	**7,788**	66,338	**79,748**	159,567	87,536
`rdfs:range`	32,338	**4,340**	37,529	**75,338**	149,545	79,678
`:hasValue`	9,903	0	82,853	0	92,756	-
`:allValuesFrom`	51,988	-	22,145	-	74,133	-
`rdfs:subPropertyOf`	3,365	**147**	22,481	**26,742**	52,734	26,888
`:maxCardinality`	26,963	-	-	-	26,963	-
`:inverseOf`	75	52	6,397	**18,363**	24,887	18,363
`:cardinality`	20,006	-	-	-	20,006	-
`:unionOf`	-	-	21,671	-	21,671	-
`:minCardinality`	15,187	-	-	-	15,187	-
`:oneOf`	-	-	6,171	-	6,171	-
`:equivalentProperty`	105	24	187	**696**	1,012	696
Class						
`:FunctionalProperty`	9,616	-	-	**18,111**	27,727	18,111
`:InverseFunctionalProperty`	872	-	-	**3,080**	3,952	3,080
`:TransitiveProperty`	807	-	-	**1,994**	2,801	1,994
`:SymmetricProperty`	265	-	-	**351**	616	351
OVERALL	2,450,020	785,661	3,365,595	1,990,035	8,591,311	2,585,708

Table 6: Authoritative analysis of T-Box statements in 1.1b dataset for each primitive where dropped statements are highlighted in bold

plateau shown at (b), the `:sameAs` equality file is closed for the first time and a local fixpoint is being calculated to derive the initial `:sameAs` statements for future rules; also during the plateau at (b), the second iteration for the `:sameAs` fixpoint (which, for the first time, consolidates the key join variables in files for rules **rdfp2**, **rdfp1′**, **rdfc4a**, **rdfc4b** according to all `:sameAs` statements produced thus far) produces 1,018 new such statements, with subsequent iterations producing 145, 2, and 0 new statements respectively.

The second pronounced step at (c) is attributable to 265k transitive inferences, followed by 1.7k symmetric-transitive inferences. The proceeding slope at (d) is caused by inferences on **rdfc3c** (`:intersectionOf` - 265 inferences) and **rdfp15′** (`:someValuesFrom` - 36k inferences), with rule **rdfp16′** (`:allValuesFrom` - 678k inferences) producing the final significant step at (e). The first complete iteration of the overall fixpoint calculation is now complete.

Since the first local `:sameAs` fixpoint, 22k mostly `rdf:type` statements have been written back to the cardinality rule files, 4 statements to the `:InverseFunctionalProperty` file and 14 to the `:FunctionalProperty` file. Thus, the `:sameAs` fixpoint is re-executed at (f), with no new statements found. The final, minor, staggered step at (g) occurs after the second `:sameAs` fixpoint when, most notably, rule **rdfp4** (`:TransitiveProperty`) produces 24k inferences, rule **rdfc3c** (`:intersectionOf`) produces 6.7k inferences, and rule **rdfp16′** (`:allValuesFrom`) produces 7.3k new statements.

The final, extended plateau at (h) is caused by rules which produce/consume `rdf:type` statements. In particular, the fixpoint encounters `:allValuesFrom` inferencing producing a minor contribution of statements ($\leq$ 2) which lead to an update and re-execution of `:allValuesFrom` inferencing and `:intersectionOf` reasoning. In particular, `:allValuesFrom` required 66 recursive iterations to reach a fixpoint. We identified the problematic data as follows:

```
@prefix veml:  <http://www.icsi.berkeley.edu/~snarayan/VEML.owl#>
@prefix verl:  <http://www.icsi.berkeley.edu/~snarayan/VERL.owl#>
@prefix data:  <http://www.icsi.berkeley.edu/~snarayan/meeting01.owl#>
...
```

# FROM veml: (T-BOX)

```
veml:sceneEvents rdfs:range veml:EventList .
veml:EventList rdfs:subClassOf _:r1 ; rdfs:subClassOf _:r2 .
_:r1 :allValuesFrom verl:Event ; :onProperty rdf:first .
_:r2 :allValuesFrom veml:EventList ; :onProperty rdf:rest .
```

# FROM data: (A-BOX)

```
data:scene veml:sceneEvents ( data:1 , ..., data:65 ) .
```

# EXAMPLE COLLECTION SNIPPET

```
_:cN rdf:first data:N ; rdf:rest _:cN+1 .
```

From the above data, each iteration of `:allValuesFrom` reasoning and subsequent subclass reasoning produced:

# INPUT TO ALL-VALUES-FROM, ITERATION 0
# FROM INPUT

```
(_:c1..._:c65) rdf:first (data:1 ...  data:65) .
```

# FROM RANGE

```
_:c1 a veml:EventList .
```

# OUTPUT ALL-VALUES-FROM, ITERATION N

```
_:dataN a verl:Event .
_:cN+1 a veml:EventList .
```

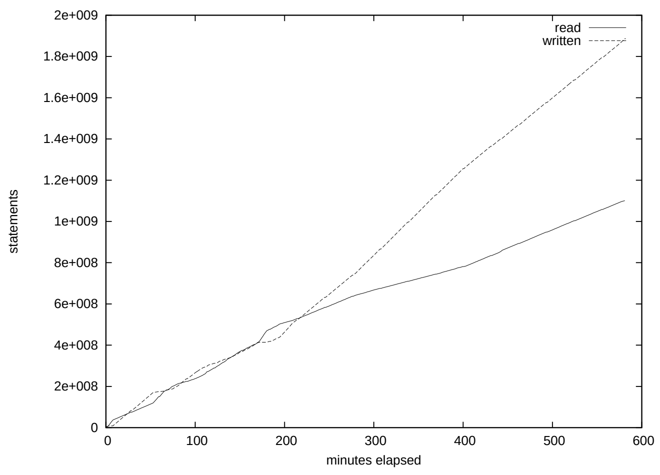


Figure 7: Performance of applying ruleset $\mathcal{R}0 \cup \mathcal{R}1$ on the 1.1b dataset

# FROM SUBCLASS ON ABOVE
# ADDED TO ALL-VALUES-FROM, ITERATION N+1
```
_:cN+1 rdf:type _:r1 ; rdf:type _:r2 .
```

In particular, a small contribution of input statements requires a merge-sort and re-scan of the file in question. This could indeed be solved by implementing binary-search lookup functionality over the sorted files for small input from a previous round; however, this would break with our initial aim of performing reasoning using only the primitives of file-scanning and multi-way merge-sort.

Finally in the reasoning process, we must perform consolidation of the input data and the output inferred statements according to the `:sameAs` index produced in the previous step. The first step involves sorting the input and inferred data according to natural SPOC order; the process took 6.4 hours and rewrote 35.4m statements into pivotal form. The second step involves subsequent sorting of the data according to inverse OPSC order; the process took 8.2 hours and rewrote 8.5m statements. The expense of these steps is primarily attributable to applying multi-way merge-sorting over all data in both sorting orders.

Although the degradation of performance related to the on-disk fixpoint computation of ruleset $\mathcal{R}2 \cup \mathcal{R}3$ is significant, if one is prepared to trade completeness (as we define it) for computational efficiency, the fixpoint calculation can be restrained to only perform a small, known amount of iterations (e.g., inferencing of the majority of statements in Figure 6 takes place over approx. 3 hours). Only minute amounts of inferred statements are produced in latter iterations of the fixpoint.
still, most inferences are produced after the initial scan which takes approx. 79 minutes. Thus, even after application of only $\mathcal{R}0$ and $\mathcal{R}1$ rules, the majority of inferencing has been conducted. This simpler more practical reasoning subset exhibits linear scale, as is visible for the first stage of Figure 5 prior to the on-disk computations. Along these lines, we present in Figure 7 the performance of applying rules $\mathcal{R}0$ and $\mathcal{R}1$ to the 1.1b statement dataset, in one scan, with respect to the T-Box derived from that dataset

as described above. In particular, we refer to the linear trend present; upon inspection, one can see that minor slow-down in the rate of statements read is attributable to an increased throughput in terms of output statements (disk write operations).

Finally, Table 7 lists the number of times each rule was fired for reasoning on the 1.1b dataset, reasoning using only $\mathcal{R}0 \cup \mathcal{R}1$ on the 147m dataset and also of applying all rules to the 147m dataset. Again, from both Figure 5 and Table 7 we can deduce that the bulk of current web reasoning is covered by those rules ($\mathcal{R}0 \cup \mathcal{R}1$) which exhibit linear scale.

Rule	1.1b - $\mathcal{R}0 - 1$	147M - $\mathcal{R}0 - 1$	147M - $\mathcal{R}0 - 3$
$\mathcal{R}0$			
rdfc0	35,157	6,084	6,084
$\mathcal{R}1$			
rdfs2	591,304,476	30,203,111	30,462,570
rdfs3′	596,661,696	31,789,905	32,048,477
rdfs7′	156,744,587	27,723,256	27,882,492
rdfs9	1,164,619,890	64,869,593	65,455,001
rdfp3′	562,426	483,204	483,204
rdfp8a′	231,661,554	9,404,319	9,556,544
rdfp8b′	231,658,162	9,404,111	9,556,336
rdfp12a′	8,153,304	23,869	38,060
rdfp12b′	57,116	17,769	25,362
rdfp13a′	5,667,464	11,478	11,478
rdfp13b′	6,642	4,350	4,350
rdfp14a′	98,601	39,422	39,902
rdfp14b′	104,780	43,886	44,390
rdfc1	15,198,615	1,492,395	1,595,293
rdfc2	584,913	337,141	337,279
rdfc3a	115,416	3,075	17,224
rdfc3b	54	8	8
$\mathcal{R}2$			
rdfp1′	-	-	31,174
rdfp2	-	-	2,097,007
rdfp4	-	-	291,048
rdfp15′	-	-	42,098
rdfp16′	-	-	685,738
rdfc3c	-	-	6,976
rdfc4a	-	-	211
rdfc4b	-	-	246

Table 7: Count of number of statements inferred for applying the given ruleset on the given dataset.

6 Related Work

OWL reasoning, specifically query answering over OWL Full, is not tackled by typical DL Reasoners; such as FaCT++ [45], RACER [19] or Pellet [40]; which focus on complex reasoning tasks such as subsumption checking and provable completeness of reasoning. Likewise, KAON2 [32], which reports better results on query answering, is limited to OWL-DL expressivity due to completeness requirements. Despite being able to deal with complex ontologies in a complete manner, these systems are not tailored for the particular challenges of processing large amounts of RDF data and particularly large A-Boxes.

Systems such as TRIPLE [39], JESS[18], or Jena[19] support rule representable RDFS or OWL fragments as we do, but only work in-memory whereas our framework is focused on conducting scalable reasoning using persistent storage.

The OWLIM [28] family of systems allows reasoning over a version of pD* using the TRREE: Triple Reasoning and Rule Entailment Engine. Besides the in-memory version SwiftOWLIM, which uses TRREE, there is also a version offering query-processing over a persistent image of the repository, BigOWLIM, which comes closest technically to our approach. In evaluation on 2 x Dual-Core 2GHz machines with 16GB of RAM, BigOWLIM is claimed to index over 1 bn triples from the LUBM benchmark [17] in just under 70 hours [1]; however, this figure includes indexing of the data for query-answering, and is not directly comparable with our results, and in any case, our reasoning approach strictly focuses on sensible reasoning for web data.

Some existing systems already implement a separation of T-Box and A-Box for scalable reasoning, where in particular, assertional data are stored in some RDBMS; e.g. DLDB [35], Minerva [48] and OntoDB [25]. Similar to our approach of reasoning over web data, [36] demonstrates reasoning over 166m triples using the DLDB system. Also like us, (and as we had previously introduced in [23]) they internally choose pivot identifiers to represent equivalent sets of individuals. However, they use the notion of perspectives to support inferencing based on T-Box data; in their experiment they manually selected nine T-Box perspectives, unlike our approach that deals with arbitrary T-Box data from the Web. Their evaluation was performed on a workstation with dual 64-bit CPUs and 10GB main memory on which they loaded 760k documents / 166m triples (14% larger than our 147m statement dataset) in about 350 hrs; however, unlike our evaluation, the total time taken includes indexing for query-answering.

In a similar approach to our authoritative analysis, [8] introduced restrictions for accepting sub-class and equivalent-class statements from third-party sources; they follow similar arguments to that made in this paper. However, their notion of what we call authoritativeness is based on hostnames and does not consider redirects; we argue that in both cases, e.g., use of PURL services[20] is not properly supported: (i) all documents using the same service (and having the same namespace hostname) would be 'authoritative' for each other, (ii) the document cannot be served directly by the namespace location, but only through a redirect. Indeed, further work presented in [7] introduced the notion of an *authoritative description* which is very similar to ours. In any case, we provide much more extensive treatment of the issue, supporting a much more varied range of RDF(S)/OWL constructs.

One promising alternative to authoritative reasoning for the Web is the notion of "context-dependant" or "quarantined reasoning" introduced in [11], whereby inference results are only considered valid within the given context of a document. As opposed to our approach whereby we construct one authoritative model for all web data, their approach uses a unique model for each document, based on implicit and explicit imports

[18]`http://herzberg.ca.sandia.gov/`
[19]`http://jena.sourceforge.net/`
[20]`http://purl.org/`

of the document; thus, they would infer statements within the local context which we would consider to be non-authoritative. However, they would miss inferences which can only be conducted by considering a merge of documents, such as transitive closure or equality inferences based on inverse-functional properties over multiple documents. Their evaluation was completed on three machines with quad-core 2.33GHz and 8GB main memory; they claimed to be able to load, on average, 40 documents per second.

7 Conclusion and Future Work

We have presented SAOR: a system for performing reasoning over web data based on primitives known to scale: file-scan and sorting. We maintain a separate optimised T-Box index for our reasoning procedure. To keep the resulting knowledge-base manageable, both in size and quality, we made the following modifications to traditional reasoning procedures:

- only consider a positive fragment of OWL reasoning;
- analyse the authority of sources to counter ontology hijacking;
- use pivot identifiers instead of full materialisation of equality.

We show in our evaluation that naïve inferencing over web data leads to an explosion of materialised statements and show how to prevent this explosion through analysis of the authority of data sources. We also present metrics relating to the most productive rules with regards inferencing on the Web.

Although SAOR is currently not optimised for reaching full closure, we show that our system is suitable for optimised computation of the approximate closure of a web knowledge-base w.r.t. the most commonly used RDF(S) and OWL constructs. In our evaluation, we showed that the bulk of inferencing on web data can be completed with two scans of an unsorted web-crawl.

Future work includes investigating possible distribution methods: indeed, by limiting our tool-box to file scans and sorts, our system can be implemented on multiple machines, as-is, according to known distribution methods for our foundational operations.

References

[1] Bigowlim: System doc., Oct. 2006. `http://www.ontotext.com/owlim/big/BigOWLIMSysDoc.pdf`.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-ref/`.

[3] S. Bechhofer and R. Volz. Patching syntax in owl ontologies. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 668–682. Springer, November 2004.

[4] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-schema/`.

[5] D. Brickley and L. Miller. FOAF Vocabulary Specification 0.91, Nov. 2007. `http://xmlns.com/foaf/spec/`.

[6] J. d. Bruijn and S. Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *6th International Semantic Web Conference*, number 4825 in LNCS, pages 86–99, Busan, Korea, Nov 2007.

[7] G. Cheng, W. Ge, H. Wu, and Y. Qu. Searching semantic web objects based on class hierarchies. In *Proceedings of Linked Data on the Web Workshop*, 2008.

[8] G. Cheng and Y. Qu. Term dependence on the semantic web. In *International Semantic Web Conference*, pages 665–680, oct 2008.

[9] J. de Bruijn. *Semantic Web Language Layering with Ontologies, Rules, and Meta-Modeling*. PhD thesis, University of Innsbruck, 2008.

[10] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL^{-}. Final draft d20.1v0.2, WSML, 2005.

[11] R. Delbru, A. Polleres, G. Tummarello, and S. Decker. Context dependent reasoning for semantic documents in Sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, October 2008.

[12] D. Fensel and F. van Harmelen. Unifying reasoning and search to web scale. *IEEE Internet Computing*, 11(2):96, 94–95, 2007.

[13] S. Ghilardi, C. Lutz, and F. Wolter. Did i damage my ontology? a case for conservative extensions in description logics. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 187–197, June 2006.

[14] B. C. Grau, I. Horrocks, B. Parsia, P. Patel-Schneider, and U. Sattler. Next steps for OWL. In *OWL: Experiences and Directions Workshop*, Nov. 2006.

[15] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *13th International Conference on World Wide Web*, 2004.

[16] R. V. Guha, R. McCool, and R. Fikes. Contexts for the semantic web. In *Third International Semantic Web Conference*, pages 32–46, November 2004.

[17] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[18] C. Gutiérrez, C. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Paris*, June 2004.

[19] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *International Workshop on Evaluation of Ontology-based Tools*, 2003.

[20] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *3rd Latin American Web Congress*, pages 71–80. IEEE Press, 2005.

[21] A. Harth, J. Umbrich, and S. Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *5th International Semantic Web Conference*, pages 258–271, 2006.

[22] P. Hayes. RDF Semantics. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-mt/`.

[23] A. Hogan, A. Harth, and S. Decker. Performing object consolidation on the semantic web data graph. In *1st I3 Workshop: Identity, Identifiers, Identification Workshop*, 2007.

[24] A. Hogan, A. Harth, and A. Polleres. SAOR: Authoritative Reasoning for the Web. In *Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008)*, Bankok, Thailand, Dec. 2008.

[25] D. Hondjack, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, pages 497–508, April 2007.

[26] I. Horrocks and P. F. Patel-Schneider. Reducing owl entailment to description logic satisfiability. *Journal of Web Semamtics*, 1(4):345–357, 2004.

[27] E. Jiménez-Ruiz, B. C. Grau, U. Sattler, T. Schneider, and R. B. Llavori. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proceedings of the 21st International Workshop on Description Logics (DL2008)*, May 2008.

[28] A. Kiryakov, D. Ognyanov, and D. Manov. Owlim - a pragmatic semantic repository for owl. In *Web Information Systems Engineering Workshops*, LNCS, pages 182–192, New York, USA, Nov 2005.

[29] D. Kunkle and G. Cooperman. Solving rubik's cube: disk is the new ram. *Communications of the ACM*, 51(4):31–33, 2008.

[30] J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.

[31] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 453–458, January 2007.

[32] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Forschungszentrum Informatik, Karlsruhe, Germany, 2006.

[33] B. Motik. On the properties of metamodeling in owl. *Journal of Logic and Computation*, 17(4):617–637, 2007.

[34] S. Muñoz, J. Pérez, and C. Gutiérrez. Minimal deductive systems for RDF. In *ESWC*, pages 53–67, 2007.

[35] Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. In *PSSS1 - Practical and Scalable Semantic Systems, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, October 2003.

[36] Z. Pan, A. Qasem, S. Kanitkar, F. Prabhakar, and J. Heflin. Hawkeye: A practical large scale demonstration of semantic web integration. In *OTM Workshops (2)*, volume 4806 of *Lecture Notes in Computer Science*, pages 1115–1124. Springer, November 2007.

[37] P. F. Patel-Schneider and I. Horrocks. Owl web ontology language semantics and abstract syntax section 4. mapping to rdf graphs. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-semantics/mapping.html`.

[38] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, Jan. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[39] M. Sintek and S. Decker. Triple - a query, inference, and transformation language for the semantic web. In *1st International Semantic Web Conference*, pages 364–378, 2002.

[40] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[41] M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-guide/`.

[42] H. J. ter Horst. Combining rdf and part of owl with rules: Semantics, decidability, complexity. In *4th International Semantic Web Conference*, pages 668–684, 2005.

[43] H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema ans a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3:79–115, 2005.

[44] Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Proceedings of the Fourth International Semantic Web Conference*, pages 685–701, November 2005.

[45] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *International Joint Conf. on Automated Reasoning*, pages 292–297, 2006.

[46] T. D. Wang, B. Parsia, and J. A. Hendler. A survey of the web ontology landscape. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, pages 682–694, Athens, GA, USA, Nov. 2006.

[47] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *24th International Conference on Data Engineering*. IEEE, 2008. To appear.

[48] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable owl ontology storage and inference system. In *Proceedings of The First Asian Semantic Web Conference (ASWC)*, pages 429–443, September 2006.

Published in of the 5th European Semantic Web Conference (ESWC2008), pp. 432–447, Nov. 2007, Springer LNCS vol. 3803, ext. version published as tech. report, cf. `http://www.deri.ie/fileadmin/documents/TRs/DERI-TR-2007-12-14.pdf` and as W3C member submission, cf. `http://www.w3.org/Submission/2009/01/`

XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage*

Waseem Akhtar[†] *Jacek Kopecký*[‡] *Thomas Krennwallner*[†] [§]
Axel Polleres[†]

[†] *Digital Enterprise Research Institute, National University of Ireland, Lower Dangan, Galway, Ireland*
[‡] *STI Innsbruck, University of Innsbruck, Austria. Technikerstraße 21a, 6020 Innsbruck, Austria.*
[§] *Institut für Informationssysteme 184/3, Technische Universität Wien, Austria Favoritenstraße 9-11, A-1040 Vienna, Austria.*

Abstract

With currently available tools and languages, translating between an existing XML format and RDF is a tedious and error-prone task. The importance of this problem is acknowledged by the W3C GRDDL working group who faces the issue of extracting RDF data out of existing HTML or XML files, as well as by the Web service community around SAWSDL, who need to perform lowering and lifting between RDF data from a semantic client and XML messages for a Web service. However, at the moment, both these groups rely solely on XSLT transformations between RDF/XML and the respective other XML format at hand. In this paper, we propose a more natural approach for such transformations based on merging XQuery and SPARQL into the novel language XSPARQL. We demonstrate that XSPARQL provides concise and intuitive solutions for mapping between XML and RDF in either direction, addressing both the use cases of GRDDL and SAWSDL. We also provide and describe an initial implementation of an XSPARQL engine, available for user evaluation.

1 Introduction

There is a gap within the Web of data: on one side, XML provides a popular format for data exchange with a rapidly increasing amount of semi-structured data available. On the other side, the Semantic Web builds on data represented in RDF, which is optimized

*This material is based upon works supported by the European FP6 projects inContext (IST-034718) and TripCom (IST-4-027324-STP), and by Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

```
@prefix alice:  <alice/> .
@prefix foaf:  <...foaf/0.1/> .

alice:me a foaf:Person.
alice:me foaf:knows _:c.
_:c a foaf:Person.
_:c foaf:name "Charles".
```

(a)

```
<rdf:RDF xmlns:foaf="...foaf/0.1/"
  xmlns:rdf="...rdf-syntax-ns#">
 <foaf:Person rdf:about="alice/me">
  <foaf:knows>
   <foaf:Person foaf:name="Charles"/>
  </foaf:knows>
 </foaf:Person>
</rdf:RDF>
```

(b)

```
<rdf:RDF xmlns:foaf="...foaf/0.1/"
  xmlns:rdf="...rdf-syntax-ns#">
 <rdf:Description rdf:nodeID="x">
  <rdf:type
      rdf:resource=".../Person"/>
  <foaf:name>Charles</foaf:name>
 </rdf:Description>
 <rdf:Description
      rdf:about="alice/me">
  <rdf:type
      rdf:resource=".../Person"/>
  <foaf:knows rdf:nodeID="x"/>
 </rdf:Description>
</rdf:RDF>
```

(c)

```
<rdf:RDF xmlns:foaf="...foaf/0.1/"
  xmlns:rdf="...rdf-syntax-ns#">
 <rdf:Description rdf:about="alice/me">
  <foaf:knows rdf:nodeID="x"/>
 </rdf:Description>
 <rdf:Description rdf:about="alice/me">
  <rdf:type rdf:resource=".../Person"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="x">
  <foaf:name>Charles</foaf:name>
 </rdf:Description>
 <rdf:Description rdf:nodeID="x">
  <rdf:type rdf:resource=".../Person"/>
 </rdf:Description>
</rdf:RDF>
```

(d)

Figure 1: Different representations of the same RDF graph

for data interlinking and merging; the amount of RDF data published on the Web is also increasing, but not yet at the same pace. It would clearly be useful to enable reuse of XML data in the RDF world and vice versa. However, with currently available tools and languages, translating between XML and RDF is not a simple task.

The importance of this issue is currently being acknowledged within the W3C in several efforts. The Gleaning Resource Descriptions from Dialects of Languages [11] (GRDDL) working group faces the issue of extracting RDF data out of existing (X)HTML Web pages. In the Semantic Web Services community, RDF-based client software needs to communicate with XML-based Web services, thus it needs to perform transformations between its RDF data and the XML messages that are exchanged with the Web services. The Semantic Annotations for WSDL (SAWSDL) working group calls these transformations *lifting* and *lowering* (see [14, 16]). However, both these groups propose solutions which rely solely on XSL transformations (XSLT) [12] between RDF/XML [2] and the respective other XML format at hand. Using XSLT for handling RDF data is greatly complicated by the flexibility of the RDF/XML format. XSLT (and XPath) were optimized to handle XML data with a simple and known hierarchical structure, whereas RDF is conceptually different, abstracting away from fixed, tree-like structures. In fact, RDF/XML provides a lot of flexibility in how RDF graphs can be serialized. Thus, processors that handle RDF/XML as XML data (not as a set of triples) need to take different possible representations into account when looking for pieces of data. This is best illustrated by a concrete example: Fig. 1 shows four versions of the same FOAF (cf. `http://www.foaf-project.org`) data.[1] The first version uses Turtle [3], a simple and readable textual format for RDF, inaccessible to pure XML processing tools though; the other three versions are all RDF/XML, ranging from concise (b) to verbose (d).

The three RDF/XML variants look very different to XML tools, yet exactly the same to RDF tools. For any variant we could create simple XPath expressions that

[1]In listings and figures we often abbreviate well-known namespace URIs (`http://www.w3.org/1999/02/22-rdf-syntax-ns#`, `http://xmlns.com/foaf/0.1/`, etc.) with "...".

extract for instance the names of the persons known to Alice, but a single expression that would correctly work in all the possible variants would become more involved. Here is a list of particular features of the RDF data model and RDF/XML syntax that complicate XPath+XSLT processing:

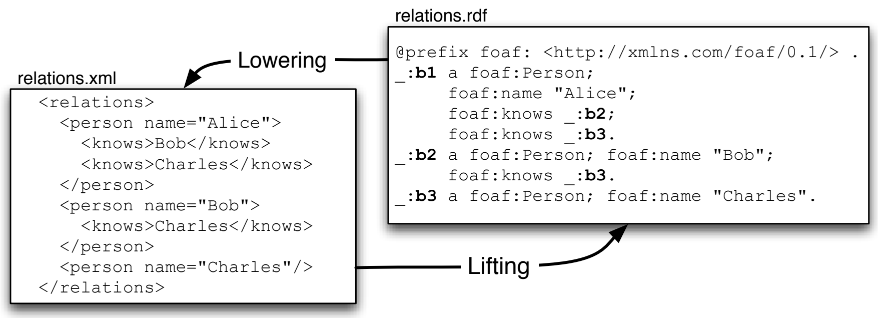
- Elements denoting properties can directly contain value(s) as nested XML, or reference other descriptions via the `rdf:resource` or `rdf:nodeID` attributes.
- References to resources can be relative or absolute URIs.
- Container membership may be expressed as `rdf:li` or `rdf:_1`, `rdf:_2`, etc.
- Statements about the same subject do not need to be grouped in a single element.
- String-valued property values such as `foaf:name` in our example (and also values of `rdf:type`) may be represented by XML element content or as attribute values.
- The type of a resource can be represented directly as an XML element name, with an explicit `rdf:type` XML element, or even with an `rdf:type` attribute.

This is not even a complete list of the issues that complicate the formulation of adequate XPath expressions that cater for every possible alternative in how one and the same RDF data might be structured in its concrete RDF/XML representation.

Apart from that, simple reasoning (e.g., RDFS materialization) improves data queries when accessing RDF data. For instance, in FOAF, every Person (and Group and Organization etc.) is also an Agent, therefore we should be able to select all the instances of `foaf:Agent`. If we wanted to write such a query in XPath+XSLT, we literally would need to implement an RDFS inference engine within XSLT. Given the availability of RDF tools and engines, this seems to be a dispensable exercise.

Recently, two new languages have entered the stage for processing XML and RDF data: XQuery [7] is a W3C Recommendation since early last year and SPARQL [22] has finally received W3C's Recommendation stamp in January 2008. While both languages operate in their own worlds – SPARQL in the RDF- and XQuery in the XML-world – we show in this paper that the merge of both in the novel language XSPARQL has the potential to finally bring XML and RDF closer together. XSPARQL provides concise and intuitive solutions for mapping between XML and RDF in either direction, addressing both the use cases of GRDDL and SAWSDL. As a side effect, XSPARQL may also be used for RDF to RDF transformations beyond the capabilities of "pure" SPARQL. We also describe an implementation of XSPARQL, available for user evaluation.

In the following, we elaborate a bit more in depth on the use cases of lifting and lowering in the contexts of both GRDDL and SAWSDL in Section 2 and discuss how they can be addressed by XSLT alone. Next, in Section 3 we describe the two starting points for an improved lifting and lowering language – XQuery and SPARQL – before we announce their happy marriage to XSPARQL in Section 4. Particularly, we extend XQuery's **FLWOR** expressions with a way of iterating over SPARQL results. We define the semantics of XSPARQL based on the XQuery semantics in [9], and describe a rewriting algorithm that translates XSPARQL to XQuery. By this we can show that XSPARQL is a conservative extension of both XQuery and SPARQL. We wrap up



```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
     foaf:name "Alice";
     foaf:knows _:b2;
     foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```

Figure 2: From XML to RDF and back: “lifting” and “lowering”

the paper with an outlook to related and future works and conclusions to be drawn in Section 5 and 6.

2 Motivation – Lifting and Lowering

As a running example throughout this paper we use a mapping between FOAF data and a customized XML format as shown in Fig. 2. The task here in either direction is to extract for all persons the names of people they know. In order to keep things simple, we use element and attribute names corresponding to the respective classes and properties in the FOAF vocabulary (i.e., `Person`, `knows`, and `name`). We assume that names in our XML file uniquely identify a person which actually complicates the transformation from XML to RDF, since we need to create a unique, distinct blank node per name. The example data is a slight variant of the data from Fig. 1, where Alice knows both Bob and Charles, Bob knows Charles, and all parties are identified by blank nodes.

Because semantic data in RDF is on a higher level of abstraction than semi-structured XML data, the translation from XML to RDF is often called “lifting” while the opposite direction is called “lowering,” as also shown in Fig. 2.

Lifting in GRDDL.

The W3C Gleaning Resource Descriptions from Dialects of Languages (GRDDL) working group has the goal to complement the concrete RDF/XML syntax with a mechanism to relate to other XML dialects (especially XHTML or “microformats”) [11]. GRDDL focuses on the lifting task, i.e., extracting RDF from XML. To this end, the working group recently published a finished Recommendation which specifies how XML files or XML Schema namespace documents can reference transformations that are then processed by a GRDDL-aware application to extract RDF from the respective source file. Typically – due to its wide support – XSLT [12] is the language of choice to describe such transformations. However, writing XSLT can be cumbersome, since it is a general-purpose language for producing XML without special support for creating RDF. For our running example, the XSLT in Fig. 3(a) could be used to generate RDF/XML from the relations.xml file in Fig. 2 in an attempt to solve the lifting step. Using GRDDL, we can link this XSLT file mygrddl.xsl from relations.xml by changing the root element of the latter to:

```
<xsl:stylesheet
  xmlns:xsl="...XSL/Transform"
  xmlns:foaf="...foaf/0.1/"
  xmlns:rdf="...rdf-syntax-ns#"
  version="2.0">

<xsl:template match="/relations">
  <rdf:RDF>
    <xsl:apply-templates />
  </rdf:RDF>
</xsl:template>

<xsl:template match="person">
  <foaf:Person>
    <foaf:name>
      <xsl:value-of
          select="./@name"/>
    </foaf:name>
    <xsl:apply-templates/>
  </foaf:Person>
</xsl:template>

<xsl:template match="knows">
  <foaf:knows><foaf:Person>
    <foaf:name>
      <xsl:apply-templates/>
      </foaf:name>
  </foaf:Person></foaf:knows>
</xsl:template>

</xsl:stylesheet>
```

(a) mygrddl.xsl

```
<rdf:RDF xmlns:rdf="...rdf-syntax-ns#"
         xmlns:foaf="...foaf/0.1/">
<foaf:Person>
  <foaf:name>Alice</foaf:name>
  <foaf:knows><foaf:Person>
      <foaf:name>Bob</foaf:name>
  </foaf:Person></foaf:knows>
  <foaf:knows><foaf:Person>
      <foaf:name>Charles</foaf:name>
  </foaf:Person></foaf:knows>
</foaf:Person>
<foaf:Person>
  <foaf:name>Bob</foaf:name>
  <foaf:knows><foaf:Person>
      <foaf:name>Charles</foaf:name>
  </foaf:Person></foaf:knows>
</foaf:Person>
<foaf:Person>
  <foaf:name>Charles</foaf:name>
</foaf:Person>
</rdf:RDF>
```

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/>.
_:b1 a foaf:Person; foaf:name "Alice";
     foaf:knows _:b2; foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob".
_:b3 a foaf:Person; foaf:name "Charles".
_:b4 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b5 .
_:b5 a foaf:Person; foaf:name "Charles" .
_:b6 a foaf:Person; foaf:name "Charles".
```

(b) Result of the GRDDL transform
in RDF/XML (up) and Turtle (down)

Figure 3: Lifting attempt by XSLT

```
<relations xmlns:grddl="http://www.w3.org/2003/g/data-view#"
           grddl:transformation="mygrddl.xsl"> ...
```

The RDF/XML result of the GRDDL transformation is shown in the upper part of Fig. 3(b). However, if we take a look at the Turtle version of this result in the lower part of Fig. 3(b) and compare it with the relations.rdf file in Fig. 2 we see that this transformation creates too many blank nodes, since this simple XSLT does not merge equal names into the same blank nodes.

XSLT is a Turing-complete language, and theoretically any conceivable transformation can be programmed in XSLT; so, we could come up with a more involved stylesheet that creates unique blank node identifiers per name to solve the lifting task as intended. However, instead of attempting to repair the stylesheet from Fig. 3(a) let us rather ask ourselves whether XSLT is the right tool for such transformations. The claim we make is that specially tailored languages for RDF-XML transformations like XSPARQL which we present in this paper might be a more suitable alternative to alleviate the drawbacks of XSLT for the task that GRDDL addresses.

Lifting/Lowering in SAWSDL.

While GRDDL is mainly concerned with lifting, in SAWSDL (Semantic Annotations for WSDL and XML Schema) there is a strong need for translations in the other direction as well, i.e., from RDF to arbitrary XML.

SAWSDL is the first standardized specification for semantic description of Web services. Semantic Web Services (SWS) research aims to automate tasks involved in

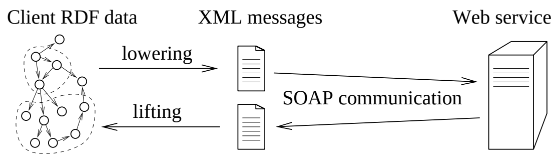


Figure 4: RDF data lifting and lowering for WS communication

the use of Web services, such as service discovery, composition and invocation. However, SAWSDL is only a first step, offering hooks for attaching semantics to WSDL components such as operations, inputs and outputs, etc. Eventually, SWS shall enable client software agents or services to automatically communicate with other services by means of semantic mediation on the RDF level. The communication requires both lowering and lifting transformations, as illustrated in Fig. 4. Lowering is used to create the request XML messages from the RDF data available to the client, and lifting extracts RDF from the incoming response messages.

As opposed to GRDDL, which provides hooks to link XSLT transformations on the level of whole XML or namespace documents, SAWSDL provides a more fine-grained mechanism for "semantic adornments" of XML Schemas. In WSDL, schemata are used to describe the input and output messages of Web service operations, and SAWSDL can annotate messages or parts of them with pointers to relevant semantic concepts plus links to lifting and lowering transformations. These links are created using the `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping` attributes which reference the transformations within XSL elements (`xsl:element`, `xsl:attribute`, etc.) describing the respective message parts.

SAWSDL's schema annotations for lifting and lowering are not only useful for communication with web services from an RDF-aware client, but for service mediation in general. This means that the output of a service S_1 uses a different message format than service S_2 expects as input, but it could still be used if services S_1 and S_2 provide lifting and lowering schema mappings, respectively, which map from/to the same ontology, or, respectively, ontologies that can be aligned via ontology mediation techniques (see [13]).

Lifting is analogous to the GRDDL situation – the client or an intermediate mediation service receives XML and needs to extract RDF from it –, but let us focus on RDF data lowering now. To stay within the boundaries of our running example, we assume a social network site with a Web service for querying and updating the list of a user's friends. The service accepts an XML format à la relations.xml (Fig. 2) as the message format for updating a user's (client) list of friends.

Assuming the client stores his FOAF data (relations.rdf in Fig. 2) in RDF/XML in the style of Fig. 1(b), the simple XSLT stylesheet mylowering.xsl in Fig. 5 would perform the lowering task. The service could advertise this transformation in its SAWSDL by linking mylowering.xsl in the `sawsdl:loweringSchemaMapping` attribute of the XML Schema definition of the `relations` element that conveys the message payload. However, this XSLT will break if the input RDF is in any other variant shown in Fig. 1. We could create a specific stylesheet for each of the presented variants, but creating one that handles all the possible RDF/XML forms would be much more complicated.

In recognition of this problem, SAWSDL contains a non-normative example which performs a lowering transformation as a sequence of a SPARQL query followed by

```
<xsl:stylesheet version="1.0" xmlns:rdf="...rdf-syntax-ns#"
  xmlns:foaf="...foaf/0.1/" xmlns:xsl="...XSL/Transform">
<xsl:template match="/rdf:RDF">
  <relations><xsl:apply-templates select=".//foaf:Person"/></relations>
</xsl:template>
<xsl:template match="foaf:Person"><person name="./@foaf:name">
  <xsl:apply-templates select="./foaf:knows"/>
</person></xsl:template>
<xsl:template match="foaf:knows[@rdf:nodeID]"><knows>
   <xsl:value-of select="//foaf:Person[@rdf:nodeID=./@rdf:nodeID]/@foaf:name"/>
</knows></xsl:template>
<xsl:template match="foaf:knows[foaf:Person]">
  <knows><xsl:value-of select="./foaf:Person/@foaf:name"/></knows>
</xsl:template>
</xsl:stylesheet>
```

Figure 5: Lowering attempt by XSLT (mylowering.xsl)

an XSLT transformation on SPARQL's query results XML format [8]. Unlike XSLT or XPath, SPARQL treats all the RDF input data from Fig. 1 as equal. This approach makes a step in the right direction, combining SPARQL with XML technologies. The detour through SPARQL's XML query results format however seems to be an unnecessary burden. The XSPARQL language proposed in this paper solves this problem: it uses SPARQL pattern matching for selecting data as necessary, while allowing the construction of arbitrary XML (by using XQuery) for forming the resulting XML structures.

As more RDF data is becoming available on the Web which we want to integrate with existing XML-aware applications, SAWSDL will obviously not remain the only use case for lowering.

3 Starting Points: XQuery and SPARQL

In order to end up with a better suited language for specifying translations between XML and RDF addressing both the lifting and lowering use cases outlined above, we can build up on two main starting points: XQuery and SPARQL. Whereas the former allows a more convenient and often more concise syntax than XSLT for XML query processing and XML transformation in general, the latter is the standard for RDF querying and construction. Queries in each of the two languages can roughly be divided in two parts: (i) the retrieval part (*body*) and (ii) the result construction part (*head*). Our goal is to combine these components for both languages in a unified language, XSPARQL, where XQuery's and SPARQL's heads and bodies may be used interchangeably. Before we go into the details of this merge, let us elaborate a bit on the two constituent languages.

3.1 XQuery

As shown in Fig. 6(a) an XQuery starts with a (possibly empty) prolog (**P**) for namespace, library, function, and variable declarations, followed by so called **FLWOR** expressions, denoting body (**FLWO**) and head (**R**) of the query. We only show namespace declarations in Fig. 6 for brevity.

As for the body, **`for`** clauses (**F**) can be used to declare variables looping over the XML nodeset returned by an XPath expression. Alternatively, to bind the entire

Prolog:	P	`declare namespace` *prefix*=`"`*namespace-URI*`"`
Body:	F	`for` *var* `in` *XPath-expression*
	L	`let` *var* `:=` *XPath-expression*
	W	`where` *XPath-expression*
	O	`order by` *XPath-expression*
Head:	R	`return` *XML+ nested XQuery*

(a) Schematic view on XQuery

Prolog:	P	`prefix` *prefix*: *<namespace-URI>*
Head:	C	`construct` { *template* }
Body:	D	`from` / `from named` *<dataset-URI>*
	W	`where` { *pattern* }
	M	`order by` *expression*
		`limit` *integer* > 0
		`offset` *integer* > 0

(b) Schematic view on SPARQL

Prolog:	P	`declare namespace` *prefix*=`"`*namespace-URI*`"` or `prefix` *prefix*: *<namespace-URI>*	
Body:	F	`for` *var* `in` *XPath-expression*	
	L	`let` *var* := *XPath-expression*	
	W	`where` *XPath-expression*	
	O	`order by` *expression*	**or**
	F'	`for` *varlist*	
	D	`from` / `from named` *<dataset-URI>*	
	W	`where` { *pattern* }	
	M	`order by` *expression*	
		`limit` *integer* > 0 `offset` *integer* > 0	
Head:	C	`construct` { *template (with nested XSPARQL)* }	**or**
	R	`return` *XML+ nested XSPARQL*	

(c) Schematic view on XSPARQL

Figure 6: An overview of XQuery, SPARQL, and XSPARQL

result of an XPath query to a variable, **`let`** assignments can be used. The **`where`** part (**W**) defines an XPath condition over the current variable bindings. Processing order of results of a **`for`** can be specified via a condition in the **`order by`** clause (**O**).

In the head (**R**) arbitrary well-formed XML is allowed following the **`return`** keyword, where variables scoped in an enclosing **`for`** or **`let`** as well as nested XQuery **FLWOR** expressions are allowed. Any XPath expression in **FLWOR** expressions can again possibly involve variables defined in an enclosing **`for`** or **`let`**, or even nested XQuery **FLWOR** expressions. Together with a large catalogue of built-in functions [17], XQuery thus offers a flexible instrument for arbitrary transformations. For more details, we refer the reader to [7, 9].

The lifting task of Fig. 2 can be solved with XQuery as shown in Fig. 7(a). The resulting query is quite involved, but completely addresses the lifting task, including unique blank node generation for each person: We first select all nodes containing person names from the original file for which a blank node needs to be created in variable `$p` (line 3). Looping over these nodes, we extract the actual names from either the value of the `name` attribute or from the `knows` element in variable `$n`. Finally, we compute the position in the original XML tree as blank node identifier in variable `$id`. The **`where`** clause (lines 12–14) filters out only the last name for duplicate occurrences of the same name. The nested **`for`** (lines 19–31) to create nested `foaf:knows` elements again loops over persons, with the only differences that only those nodes are filtered out (line 25), which are known by the person with the name from the outer **`for`** loop.

While this is a valid solution for lifting, we still observe the following drawbacks: (1) We still have to build RDF/XML "manually" and cannot make use of the more readable and concise Turtle syntax; and

(2) if we had to apply XQuery for the lowering task, we still would need to cater for all kinds of different RDF/XML representations. As we will see, both these drawbacks are alleviated by adding some SPARQL to XQuery.

3.2 SPARQL

Fig. 6(b) shows a schematic overview of the building blocks that SPARQL queries consist of. Again, we do not go into details of SPARQL here (see [22, 19, 20] for

```
declare namespace foaf="...foaf/0.1/";
declare namespace rdf="...-syntax-ns#";
let $persons := //*[@name or ../knows]
return
<rdf:RDF>
 {
 for $p in $persons
 let $n := if( $p[@name] )
           then $p/@name else $p
 let $id :=count($p/preceding::*)
           +count($p/ancestor::*)
 where
  not(exists($p/following::*[
      @name=$n or data(.)=$n]))
 return
 <foaf:Person rdf:nodeId="b{$id}">
  <foaf:name>{data($n)}</foaf:name>
  {
  for $k in $persons
  let $kn := if( $k[@name] )
             then $k/@name else $k
  let $kid :=count($k/preceding::*)
             +count($k/ancestor::*)
  where
   $kn = data(//*[@name=$n]/knows) and
   not(exists($kn/../following::*[
       @name=$kn or data(.)=$kn]))
  return
  <foaf:knows>
   <foaf:Person rdf:nodeID="b{$kid}"/>
  </foaf:knows>
  }
 </foaf:Person>
 }
</rdf:RDF>
```

(a) XQuery

```
declare namespace foaf="...foaf/0.1/";
declare namespace rdf="...-syntax-ns#";
let $persons := //*[@name or ../knows]
return


 for $p in $persons
 let $n := if( $p[@name] )
           then $p/@name else $p
 let $id :=count($p/preceding::*)
           +count($p/ancestor::*)
 where
  not(exists($p/following::*[
      @name=$n or data(.)=$n]))
 construct {
 _:b{$id} a foaf:Person;
         foaf:name {data($n)}.
  {
  for $k in $persons
  let $kn := if( $k[@name] )
             then $k/@name else $k
  let $kid :=count($k/preceding::*)
             +count($k/ancestor::*)
  where
   $kn = data(//*[@name=$n]/knows) and
   not(exists($kn/../following::*[
       @name=$kn or data(.)=$kn]))
  construct {
   _:b{$id} foaf:knows _:b{$kid}.
   _:b{$kid} a foaf:Person.
   }
  }
 }
```

(b) XSPARQL

Figure 7: Lifting using XQuery and XSPARQL

formal details), since we do not aim at modifying the language, but concentrate on the overall semantics of the parts we want to reuse. Like in XQuery, namespace prefixes can be specified in the Prolog (**P**). In analogy to **FLWOR** in XQuery, let us define so-called **DWMC** expressions for SPARQL.

The body (**DWM**) offers the following features. A *dataset* (**D**), i.e., the set of source RDF graphs, is specified in **`from`** or **`from named`** clauses. The **`where`** part (**W**) – unlike XQuery – allows to match parts of the dataset by specifying a graph *pattern* possibly involving variables, which we denote $\mathrm{vars}(\textit{pattern})$. This pattern is given in a Turtle-based syntax, in the simplest case by a set of triple patterns, i.e., triples with variables. More involved patterns allow unions of graph patterns, optional matching of parts of a graph, matching of named graphs, etc. Matching patterns on the conceptual level of RDF graphs rather than on a concrete XML syntax alleviates the pain of having to deal with different RDF/XML representations; SPARQL is agnostic to the actual XML representation of the underlying source graphs. Also the RDF merge of several source graphs specified in consecutive **`from`** clauses, which would involve renaming of blank nodes at the pure XML level, comes for free in SPARQL. Finally, variable bindings matching the **`where`** pattern in the source graphs can again be ordered, but also other solution modifiers (**M**) such as **`limit`** and **`offset`** are allowed to restrict the number of solutions considered in the result.

In the head, SPARQL's **`construct`** clause (**C**) offers convenient and XML-independent means to create an output RDF graph. Since we focus here on RDF construc-

```
prefix vc:  <...vcard-rdf/3.0#>
prefix foaf:  <...foaf/0.1/>
construct {$X foaf:name $FN.}
from <vc.rdf>
where { $X vc:FN $FN .}
```

(a)

```
prefix vc:  <...vcard-rdf/3.0#>
prefix foaf:  <...foaf/0.1/>
construct {_:b foaf:name
          {fn:concat($N," ",$F)}.}
from <vc.rdf>
where { $P vc:Given $N. $P vc:Family $F.}
```

(b)

Figure 8: RDF-to-RDF mapping in SPARQL (a) and an enhanced mapping in XSPARQL (b)

```
<relations>
  {
    for $Person $Name from <relations.rdf>
    where { $Person foaf:name $Name }
    order by $Name
    return
      <person name="{$Name}">
        {
          for $FName from <relations.rdf>
          where { $Person foaf:knows $Friend.
                  $Person foaf:name $Name.
                  $Friend foaf:name $Fname }
          return <knows>{$FName}</knows>
        }
      </person>
  }
</relations>
```

Figure 9: Lowering using XSPARQL

tion, we omit the **`ask`** and **`select`** SPARQL query forms in Fig. 6(b) for brevity. A **`construct`** *template* consists of a list of triple patterns in Turtle syntax possibly involving variables, denoted by vars(*template*), that carry over bindings from the **`where`** part. SPARQL can be used as transformation language between different RDF formats, just like XSLT and XQuery for transforming between XML formats. A simple example for mapping full names from vCard/RDF (`http://www.w3.org/TR/vcard-rdf`) to `foaf:name` is given by the SPARQL query in Fig. 8(a).

Let us remark that SPARQL does not offer the generation of new values in the head which on the contrary comes for free in XQuery by offering the full range of XPath/XQuery built-in functions. For instance, the simple query in Fig. 8(b) which attempts to merge family names and given names into a single `foaf:name` is beyond SPARQL's capabilities. As we will see, XSPARQL will not only make reuse of SPARQL for transformations from and to RDF, but also aims at enhancing SPARQL itself for RDF-to-RDF transformations enabling queries like the one in Fig. 8(b).

4 XSPARQL

Conceptually, XSPARQL is a simple merge of SPARQL components into XQuery. In order to benefit from the more intuitive facilities of SPARQL in terms of RDF graph matching for retrieval of RDF data and the use of Turtle-like syntax for result construction, we syntactically add these facilities to XQuery. Fig. 6(c) shows the result of this "marriage." First of all, every native XQuery query is also an XSPARQL query. However we also allow the following modifications, extending XQuery's **FLWOR** expressions to what we call (slightly abusing nomenclature) **FLWOR'** expressions: (i) In the body we allow SPARQL-style **F'DWM** blocks alternatively to XQuery's **FLWO** blocks. The new **F'** **`for`** clause is very similar to XQuery's native **`for`** clause, but instead of assigning a single variable to the results of an XPath expression it allows the assignment of a whitespace separated list of variables (*varlist*) to the bindings for these variables obtained by evaluating the graph pattern of a SPARQL query of the form: **`select`** *varlist* **DWM**. (ii) In the head we allow to create RDF/Turtle directly using **`construct`** statements (**C**) alternatively to XQuery's native **`return`** (**R**).

These modifications allows us to reformulate the lifting query of Fig. 7(a) into its slightly more concise XSPARQL version of Fig. 7(b). The real power of XSPARQL in our example becomes apparent on the lowering part, where all of the other languages struggle. Fig. 9 shows the lowering query for our running example.

As a shortcut notation, we allow also to write "**`for`** *" in place of "**`for`** [*list of all variables appearing in the* **`where`** *clause*]"; this is also the default value for the **F'** clause whenever a SPARQL-style **`where`** clause is found and a **`for`** clause is missing. By this treatment, XSPARQL is also a syntactic superset of native SPARQL **`construct`** queries, since we additionally allow the following: (1) XQuery and SPARQL namespace declarations (**P**) may be used interchangeably; and

(2) SPARQL-style **`construct`** result forms (**R**) may appear before the retrieval part; note that we allow this syntactic sugar only for queries consisting of a single **FLWOR'** expression, with a single **`construct`** appearing right after the query prolog, as otherwise, syntactic ambiguities may arise. This feature is mainly added in order to encompass SPARQL style queries, but in principle, we expect the (**R**) part to appear in the end of a **FLWOR'** expression. This way, the queries of Fig. 8 are also syntactically valid for XSPARQL.[2]

4.1 XSPARQL Syntax

The XSPARQL syntax is – as sketched above – a simple extension of the grammar production rules in [7]. To simplify the definition of XSPARQL, we inherit the grammar productions of SPARQL [22] and XQuery [7] and add the prime symbol ($'$) to the rules which have been modified. We only have two fresh productions: `ReturnClause` and `SparqlForClause` (which loosely reflect lifting and lowering).

The basic elements of the XSPARQL syntax are the following:

```
[33]  FLWORExpr'      ::= (ForClause | LetClause | SparqlForClause)+
                          WhereClause?  OrderByClause?  ReturnClause
[33a] SparqlForClause ::= "for" ("$" VarName ("$" VarName)* | "*") DatasetClause
                          "where" GroupGraphPattern SolutionModifier
[33b] ReturnClause    ::= "return" ExprSingle | "construct" ConstructTemplate'
```

[2]In our implementation, we also allow **`select`** and **`ask`** queries, making SPARQL a real syntactic subset of XSPARQL.

`ConstructTemplate′` is defined in the same way as the production `ConstructTemplate` in SPARQL [22], but we additionally allow XSPARQL nested `FLWORExpr′` in subject, verb, and object place. These expressions need to evaluate to a valid RDF term, i.e.:

- an IRI or blank node in the subject position;
- an IRI in the predicate position;
- a literal, IRI or blank node in the object position.

To define this we use the SPARQL grammar rules as a starting point and replace the following productions:

```
[42] VarOrTerm′           ::=   Var′ | GraphTerm′ | literalConstruct
[43] VarOrIRIref′         ::=   Var′ | IRIref | iriConstruct
[44] Var′                 ::=   VAR2
[45] GraphTerm′           ::=   IRIref | RDFLiteral | NumericLiteral
                          |     BooleanLiteral | BlankNode | NIL
                          |     bnodeConstruct | iriConstruct

[42a] literalConstruct    ::=   "{" FLWORExpr′ "}"
[43a] iriConstruct        ::=   "<{" FLWORExpr′ "}>"
                          |     ("{" FLWORExpr′ "}")?  ":" ("{" FLWORExpr′ "}")?
[45a] bnodeConstruct      ::=   "_:{" FLWORExpr′ "}"
```

4.1.1 Syntactic restrictions of XSPARQL

Without loss of generality, we make two slight restrictions on variable names in XSPARQL compared with Xquery and SPARQL. Firstly, we disallow variable names that start with '_'; we need this restriction in XSPARQL to distinguish new auxiliary variables which we introduce in the semantics definition and in our rewriting algorithm and avoid ambiguity with user-defined variables. Secondly, in SPARQL-inherited parts we only allow variable names prefixed with '$' in order to be compliant with variable names as allowed in XQuery. Pure SPARQL allows variables of this form, but additionally allows '?' as a valid variable prefix.

Likewise, we also disallow other identifier names to start with '_', namely namespace prefixes, function identifiers, and also blank node identifiers, for similar considerations: In our rewriting we generate fixed namespaces (e.g. we always need the namespace prefix `_sparql_result:` associated with the namespace-URI `http://www.w3.org/2005/sparql-results#`, which when overridden by the user might create ambiguities in our rewriting. Similarly, we use underscores in our rewriting to disambiguate blank node identifiers created from **`construct`**s from those extracted from a query result, by appending '_' to the latter. As for function names, we use underscores to denote auxiliary functions defined in our rewriting algorithm, which again we do not want to be overridden by user-defined functions.

In total, we restrict the SPARQL grammar by redefining `VARNAME` to disallow leading underscores:

```
[97] VARNAME′ ::= ( PN_CHARS_U - '_' | [0-9] ) ( PN_CHARS_U | [0-9] | #x00B7 |
                                                 [#x0300-#x036F] | [#x203F-#x2040] )*
```

And likewise in the XQUERY grammar we do not allow underscores in the beginning of `NCName`s (defined in [5]), i.e. we modify:

```
[6] NCNameStartChar'  ::=  Letter
```

4.2 XSPARQL Semantics

The semantics of XSPARQL follows the formal treatment of the semantics given for XQuery [9]. We follow the notation provided there and define XSPARQL semantics by means of normalization mapping rules and inference rules.

We defined a new dynamic evaluation inference rules for a new built-in function *fs:sparql*, which evaluates SPARQL queries. Other modifications include normalization of the XSPARQL constructs to XQuery expressions. This means that we do do not need new grammar productions but only those defined in the XQuery Core syntax.

For the sake of brevity, we do note handle namespaces and base URIs here, i.e., we do not "import" namespace declarations from **`construct`** templates, and do not push down XQuery namespace declarations to SPARQL **`select`** queries. Conceptually, they can always be parsed from the XQuery document and appended where needed.

4.2.1 FLWOR' Expressions

Our XSPARQL syntax defines, together with the XQuery **FLWOR** expression, a new **`for`**-loop for iterating over SPARQL results: `SparqlForClause`. This object stands at the same level as XQuery's **`for`** and **`let`** expressions, i.e., such type of clauses are allowed to start new FLWOR' expressions, or may occur inside deeply nested XSPARQL queries.

To this end, our new normalization mapping rules $[\cdot]_{Expr'}$ inherit from the definition of XQuery's $[\cdot]_{Expr}$ mapping rules and overload some expressions to accommodate XSPARQL's new syntactic objects. The semantics of XSPARQL expressions hence stands on top of XQuery's semantics.

A single `SparqlForClause` is normalized as follows:

$$\left\llbracket \begin{array}{l} \texttt{for}\ \$\,VarName_1 \cdots \$\,VarName_n\ DatasetClause\ \texttt{where} \\ GroupGraphPattern\ SolutionModifier\ ReturnClause \end{array} \right\rrbracket_{Expr'}$$

$$==$$

$$\left\llbracket \begin{array}{l} \texttt{let}\ \$\_aux\_queryresult\ \texttt{:=} \\ \quad \left\llbracket \begin{array}{l} \$\,VarName_1 \cdots \$\,VarName_n\ DatasetClause\ \texttt{where} \\ GroupGraphPattern\ SolutionModifier \end{array} \right\rrbracket_{SparqlQuery} \\ \texttt{for}\ \$\_aux\_result\ \texttt{in}\ \$\_aux\_queryresult\texttt{//\_sparql\_result:result} \\ \quad [VarName_1]_{SparqlResult} \\ \qquad \vdots \\ \quad [VarName_n]_{SparqlResult} \\ ReturnClause \end{array} \right\rrbracket_{Expr}$$

Here, $[\cdot]_{SparqlQuery}$ and $[\cdot]_{SparqlResult}$ are auxiliary mapping rules for expanding the expressions:

$$[\$\,VarName]_{SparqlResult}$$

$$==$$

$$\begin{array}{l} \texttt{let}\ \$\_VarName\_Node\ \texttt{:=}\ \$\_aux\_result\texttt{/\_sparql\_result:binding}[\texttt{@name="}\,VarName\,\texttt{"}] \\ \texttt{let}\ \$\,VarName\ \texttt{:=}\ \texttt{data}(\$\_VarName\_Node/\star) \\ \texttt{let}\ \$\_VarName\_NodeType\ \texttt{:=}\ \texttt{name}(\$\_VarName\_Node/\star) \\ \texttt{let}\ \$\_VarName\_RDFTerm\ \texttt{:=}\ \texttt{\_rdf\_term}(\$\_VarName\_Node) \end{array}$$

and

$$\left[\!\!\left[\begin{array}{l}\$\,VarName_1 \cdots \$\,VarName_n\ DatasetClause \\ \texttt{where}\ GroupGraphPattern\ SolutionModifier\end{array}\right]\!\!\right]_{SparqlQuery}$$
$$==$$
$$fs{:}sparql\left(\left[\!\!\left[\begin{array}{l}\texttt{fn:concat("SELECT}\ \$\,VarName_1 \cdots \$\,VarName_n\ DataSetClause\ \texttt{where \{ ",} \\ \quad \texttt{fn:concat(}GroupGraphPattern\texttt{), " \}}\ SolutionModifier\texttt{")}\end{array}\right]\!\!\right]_{Expr'}\right)$$

The `_rdf_term`($_*VarName_Node*) function is defined in the following way:

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash \begin{array}{c}[\![\texttt{\_rdf\_term}(\$\_VarName\_Node)]\!]_{Expr} = \\ \left[\!\!\left[\begin{array}{c}\texttt{if (}\$\_VarName\_NodeType\texttt{="literal") then fn:concat("""",}\$\,VarName\texttt{,"""")} \\ \texttt{else if (}\$\_VarName\_NodeType\texttt{="bnode") then fn:concat("\_:", }\$\,VarName\texttt{)} \\ \texttt{else if (}\$\_VarName\_NodeType\texttt{="uri") then fn:concat("<", }\$\,VarName\texttt{, ">")} \\ \texttt{else ""}\end{array}\right]\!\!\right]_{Expr}\end{array}}$$

We now define the meaning of *fs:sparql*. It is, following the style of [9], an abstract function which returns a SPARQL query result XML document [8] when applied a proper SPARQL **`select`** query, i.e., *fs:sparql* conforms to the XML Schema definition `http://www.w3.org/2007/SPARQL/result.xsd`:[3]

```
fs:sparql($query as xs:string) as document-node(schema-element(_sparql_result:sparql))
```

Static typing rules applies here according to the rules given in the XQuery semantics.

Since this function must be evaluated according to the SPARQL semantics, we need to get the value of *fs:sparql* in the dynamic evaluation semantics of XSPARQL.

$$\frac{\text{The built-in function } \textit{fs:sparql} \text{ applied to } Value_1 \text{ yields } Value}{\text{dynEnv} \vdash \textbf{function}\,\textit{fs:sparql}\ \textbf{with types}\ (xs{:}string)\ \textbf{on values}\ (Value_1)\ \textbf{yields}\ Value}$$

In case of error (for instance, the query string is not syntactically correct, or the *DatasetClause* cannot be accessed), *fs:sparql* issues an error:

$$\frac{Value_1 \text{ cannot be evaluated according to SPARQL semantics}}{\text{dynEnv} \vdash \textbf{function}\,\textit{fs:sparql}\ \textbf{with types}\ (xs{:}string)\ \textbf{on values}\ (Value_1)\ \textbf{yields}\ fn{:}error()}$$

The only remaining part is defining the semantics of a `GroupGraphPattern` using our extended $[\![\cdot]\!]_{Expr'}$. This mapping rule takes care that variables in scope of XSPARQL expressions are properly substituted using the evaluation mechanism of XQuery. To this end, we assume that $[\![\cdot]\!]_{Expr'}$ takes expressions in SPARQL's `Group-GraphPattern` syntax and constructs a sequence of strings and variables, by applying the auxiliary mapping rule $[\![\cdot]\!]_{VarSubst}$ to each of the graph pattern's variables. This rule looks up bound variables from the static environment and possibly replaces them to variables or to a string expression, where the value of the string is the name of the variable. This has the effect that unbound variables in `GroupGraphPattern` will be evaluated by SPARQL instead of XQuery. The statical semantics for $[\![\cdot]\!]_{VarSubst}$ is defined below using the next inference rules. They use the new judgement $\$\,VarName$ **bound**, which holds if the variable $\$\,VarName$ is bound in the current static environment.

[3]We assume that this XML schema is imported and into the `sparql_result:` namespace.

$$\frac{\text{statEnv} \vdash \$\_VarName\_RDFTerm\ \textbf{bound}}{\text{statEnv} \vdash [\$\,VarName]_{VarSubst} = [\$\_VarName\_RDFTerm]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\,VarName\ \textbf{bound} \qquad \text{statEnv} \vdash \textbf{not}(\$\_VarName\_RDFTerm\ \textbf{bound})}{\text{statEnv} \vdash [\$\,VarName]_{VarSubst} = [\$\,VarName]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \textbf{not}(\$\,VarName\ \textbf{bound}) \qquad \text{statEnv} \vdash \textbf{not}(\$\_VarName\_RDFTerm\ \textbf{bound})}{\text{statEnv} \vdash [\$\,VarName]_{VarSubst} = [\texttt{"}\$\,VarName\texttt{"}]_{Expr}}$$

Next, we define the normalization of **for** expressions. In order to handle blank nodes appropriately in **construct** expressions, we need to decorate the variables of standard XQuery **for**-expressions with position variables. First, we must normalize **for**-expressions to core **for**-loops:

$$\left[\!\!\left[\begin{array}{l}\texttt{for}\ \$\,VarName_1\ OptTypeDeclaration_1\ OptPositionalVar_1\ \texttt{in}\ Expr_1, \cdots,\\ \$\,VarName_n\ OptTypeDeclaration_n\ OptPositionalVar_n\ \texttt{in}\ Expr_n\ ReturnClause\end{array}\right]\!\!\right]_{Expr'}$$
$$==$$
$$\left[\!\!\left[\begin{array}{l}\texttt{for}\ \$\,VarName_1\ OptTypeDeclaration_1\ OptPositionalVar_1\ \texttt{in}\ Expr_1\ \texttt{return}\\ \vdots\\ \texttt{for}\ \$\,VarName_n\ \ OptTypeDeclaration_n\ OptPositionalVar_n\ \texttt{in}\ Expr_n\\ ReturnClause\end{array}\right]\!\!\right]_{Expr'}$$

Now we can apply our decoration of the core **for**-loops (without position variables) recursively:

$$[\ \texttt{for}\ \$\,VarName_i\ OptTypeDeclaration_i\ \texttt{in}\ Expr_i\ ReturnClause\ ]_{Expr'}$$
$$==$$
$$[\ \texttt{for}\ \$\,VarName_i\ OptTypeDeclaration_i\ \texttt{at}\ \$\_VarName_i\_Pos\ \texttt{in}\ [Expr_i]_{Expr'}\ [ReturnClause]_{Expr'}\ ]_{Expr}$$

Similarly, **let** expressions must be normalized as follows:

$$\left[\!\!\left[\begin{array}{l}\texttt{let}\ \$\,VarName_1\ OptTypeDeclaration_1 := Expr_1, \cdots,\\ \$\,VarName_n\ OptTypeDeclaration_n := Expr_n\ ReturnClause\end{array}\right]\!\!\right]_{Expr'}$$
$$==$$
$$\left[\!\!\left[\begin{array}{l}\texttt{let}\ \$\,VarName_1\ OptTypeDeclaration_1 := Expr_1\ \texttt{return}\\ \vdots\\ \texttt{let}\ \$\,VarName_n\ \ OptTypeDeclaration_n := Expr_n\\ ReturnClause\end{array}\right]\!\!\right]_{Expr'}$$

Now we can recursively apply $[\cdot]_{Expr'}$ on the core **let**-expressions:

$$[\ \texttt{let}\ \$\,VarName_i\ OptTypeDeclaration_i := Expr_i\ ReturnClause\ ]_{Expr'}$$
$$==$$
$$[\ \texttt{let}\ \$\,VarName_i\ OptTypeDeclaration_i := [Expr_i]_{Expr'}\ [ReturnClause]_{Expr'}\ ]_{Expr}$$

We do not specify **where** and **order by** clauses here, as they can be handled similarly as above **let** and **for** expressions.

4.2.2 CONSTRUCT Expressions

We define now the semantics for the `ReturnClause`. Expressions of form **`return`** *Expr* are evaluated as defined in the XQuery semantics. Stand-alone **`construct`**-clauses are normalized as follows:

$$\begin{array}{c}
[\![\texttt{construct}\ ConstructTemplate']\!]_{Expr'} \\
== \\
\left[\!\!\left[\texttt{return}\ (\ [\![ConstructTemplate']\!]_{SubjPredObjList}\ )\right]\!\!\right]_{Expr}
\end{array}$$

The auxiliary mapping rule $[\cdot]_{SubjPredObjlist}$ rewrites variables and blank nodes inside of `ConstructTemplate′`s using the normalization mapping rules $[\cdot]_{Subject}$, $[\cdot]_{PredObjList}$, and $[\cdot]_{ObjList}$. They use the judgements *expr* **valid subject**, **valid predicate**, and **valid object**, which holds if the expression $expr$ is, according to the SPARQL **`construct`** syntax, a valid subject, predicate, and object, resp: i.e., subjects must be bound and not literals, predicates, must be bound, not literals and not blank nodes, and objects must be bound. If, for any reason, one criterion fails, the triple containing the ill-formed expression will be removed from the output. Free variables in the **`construct`** are unbound, hence triples containing such variables must be removed too. The boundedness condition can be checked at runtime by wrapping each variable and **FLWOR'** into a *fn:empty()* assertion, which removes the corresponding triple from the `ConstructTemplate` output.

Next we define the semantics of *validSubject*, *validPredicate* and *validObject*:

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash \begin{array}{c}[\texttt{validSubject}(\$\_VarName\_Node)]_{Expr} = \\ [\ \texttt{if}\ (validBnode(\$\_VarName\_Node)\ or\ validUri(\$\_VarName\_Node))\ \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\ ]_{Expr}\end{array}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash \begin{array}{c}[\texttt{validPredicate}(\$\_VarName\_Node)]_{Expr} = \\ [\ \texttt{if}\ (validUri(\$\_VarName\_Node))\ \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\ ]_{Expr}\end{array}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash \begin{array}{c}[\texttt{validObject}(\$\_VarName\_Node)]_{Expr} = \\ \left[\!\!\left[\begin{array}{c}\texttt{if}\ (validBnode(\$\_VarName\_Node)\ or\ validLiteral(\$\_VarName\_Node)\ or\ validUri(\$\_VarName\_Node)) \\ \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\end{array}\right]\!\!\right]_{Expr}\end{array}}$$

The definitions for *validBnode*, *validUri* and *validLiteral* are the following:

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\texttt{validBnode}(\$VarName)]_{Expr} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (\$\_VarName\_NodeType = \texttt{"blank"}) \\ \ \texttt{then}\ fn{:}true()\ \texttt{else} \\ \quad \texttt{if}\ (fn{:}matches(\$VarName, \texttt{"^\_:[a-z]([a-z|0-9|\_])*\$"}) \\ \quad \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\end{array}\right]\!\!\right]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\texttt{validUri}(\$VarName)]_{Expr} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (\$\_VarName\_NodeType = \texttt{"uri"}) \\ \ \texttt{then}\ fn{:}true()\ \texttt{else} \\ \quad \texttt{if}\ (fn{:}matches(\$VarName, \texttt{"^<([^>])*>\$"})) \\ \quad \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\end{array}\right]\!\!\right]_{Expr}}$$

We follow the URI definition of the N3 syntax (according to the regular expression available at `http://www.w3.org/2000/10/swap/grammar/n3-report.html#explicituri`) as opposed to the more extensive definition in [4].

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{validLiteral}(\$VarName)]\!]_{Expr} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (fn{:}empty(\$\_VarName\_NodeType = \texttt{"literal"}))\\ \texttt{then}\ fn{:}true()\\ \quad \texttt{if}\ (fn{:}starts{-}with(\$VarName, \text{""""})\ and\\ \quad\quad fn{:}ends{-}with(\$VarName, \texttt{"'''"})\\ \quad \texttt{then}\ fn{:}true()\ \texttt{else}\ fn{:}false()\end{array}\right]\!\!\right]_{Expr}}$$

Finally, some of the normalization rules are presented; the missing rules should be clear from the context:

$$\frac{\text{statEnv} \vdash validSubject(\,VarOrTerm)}{\text{statEnv} \vdash \begin{array}{c}[\![VarOrTerm\ PropertyListNotEmpty]\!]_{SubjPredObjList} =\\ \left[\!\!\left[fn{:}concat\,(\,[\![VarOrTerm]\!]_{Subject}\ ,\ [\![PropertyListNotEmpty]\!]_{PredObjlist}\,)\right]\!\!\right]_{Expr}\end{array}}$$

$$\begin{array}{c}[\![\,[\ PropertyListNotEmpty\ ]\,]\!]_{SubjPredObjList}\\ ==\\ \text{statEnv} \vdash \left[\!\!\left[(\ \texttt{"["},\ [\![PropertyListNotEmpty]\!]_{PredObjectList}\ ,\ \texttt{"]"}\ )\right]\!\!\right]_{Expr}\end{array}$$

$$\frac{\begin{array}{c}\text{statEnv} \vdash\ Verb\ \textbf{is valid predicate}\\ \text{statEnv} \vdash\ Object_1\ \textbf{is valid object}\\ \vdots\\ \text{statEnv} \vdash\ Object_n\ \textbf{is valid object}\end{array}}{\text{statEnv} \vdash \begin{array}{c}[\![\ Verb\ Object_1, \ldots, Object_n\ .]\!]_{PredObjectList} =\\ \left[\!\!\left[(\,[\![Verb]\!]_{Expr'}\ ,\texttt{","},\ [\![Object_1]\!]_{Expr'}\ ,\texttt{","},\ \ldots,\texttt{","},\ [\![Object_n]\!]_{Expr'}\ ,\texttt{"."})\right]\!\!\right]_{Expr}\end{array}}$$

Otherwise, if one of the premises is not true, we suppress the generation of this triple. One of the negated rules is the following:

$$\frac{\text{statEnv} \vdash \textbf{not}\ (\,VarOrTerm\ \textbf{is valid subject})}{\text{statEnv} \vdash [\![VarOrTerm\ PropertyListNotEmpty]\!]_{SubjPredObjList} = [\![\texttt{""}]\!]_{Expr}}$$

The normalization for subjects, verbs, and objects according to $[\![\cdot]\!]_{Expr'}$ is similar to *GroupGraphPattern*: all variables in it will be replaced using $[\![\cdot]\!]_{VarSubst}$.

Blank nodes inside of construction templates must be treated carefully by adding position variables from surrounding **`for`** expressions. To this end, we use $[\![\cdot]\!]_{BNodeSubst}$. Since we normalize every **`for`**-loop by attaching position variables, we just need to retrieve the available position variables from the static environment. We assume a new static environment component statEnv.posVars which holds – similar to the statEnv.varType component – all in-context *positional* variables in the given static environment, that is, the variables defined in the **`at`** clause of any enclosing **`for`** loop.

$$\frac{\text{statEnv} \vdash \text{statEnv.posVars} = VarName_1\_Pos, \cdots, VarName_n\_Pos}{\text{statEnv} \vdash [\![\_{:}BNodeName]\!]_{BNodeSubst} = [\![\texttt{fn:concat}\,(\texttt{"\_:"},\ BNodeName,\ \texttt{"\_"},\texttt{"},\ VarName_1\_Pos, \cdots, VarName_n\_Pos)\ ]\!]_{Expr}}$$

4.2.3 SPARQL Filter Operators

SPARQL filter expressions in `WHERE GroupGraphPattern` are evaluated using *fs:sparql*. But we additionally allow the following functions inherited from SPARQL in XS-PARQL:

```
BOUND($A as xs:string) as xs:boolean
isIRI($A as xs:string) as xs:boolean
isBLANK($A as xs:string) as xs:boolean
isLITERAL($A as xs:string) as xs:boolean
LANG($A as xs:string) as xs:string
DATATYPE($A as xs:string) as xs:anyURI
```

The semantics of above functions is defined as follows:

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{BOUND}(\$\,VarName)]\!]_{Expr'} = [\![\texttt{if}\ (fn{:}empty(\$\_VarName\_Node))\ \texttt{then}\ fn{:}false()\ \texttt{else}\ fn{:}true()]\!]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{isIRI}(\$\,VarName)]\!]_{Expr'} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (fn{:}empty(\$\_VarName\_NodeType = \texttt{"uri"}))\\ \texttt{then}\ fn{:}false()\ \texttt{else}\ fn{:}true()\end{array}\right]\!\!\right]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{isBLANK}(\$\,VarName)]\!]_{Expr'} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (fn{:}empty(\$\_VarName\_NodeType = \texttt{"blank"}))\\ \texttt{then}\ fn{:}false()\ \texttt{else}\ fn{:}true()\end{array}\right]\!\!\right]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_NodeType\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{isLITERAL}(\$\,VarName)]\!]_{Expr'} = \left[\!\!\left[\begin{array}{l}\texttt{if}\ (fn{:}empty(\$\_VarName\_NodeType = \texttt{"literal"}))\\ \texttt{then}\ fn{:}false()\ \texttt{else}\ fn{:}true()\end{array}\right]\!\!\right]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{LANG}(\$\,VarName)]\!]_{Expr'} = [\![fn{:}string(\$\_VarName\_Node/@xml{:}lang)]\!]_{Expr}}$$

$$\frac{\text{statEnv} \vdash \$\_VarName\_Node\ \textbf{bound}}{\text{statEnv} \vdash [\![\texttt{DATATYPE}(\$\,VarName)]\!]_{Expr'} = [\![\$\_VarName\_Node/@datatype]\!]_{Expr}}$$

4.3 Implementation

As we have seen above, XSPARQL syntactically subsumes both XQuery and SPARQL. Concerning semantics, XSPARQL equally builds on top of its constituent languages. We have extended the formal semantics of XQuery [9] by additional rules which reduce each XSPARQL query to XQuery expressions; the resulting **FLWOR**s operate on the answers of SPARQL queries in the SPARQL XML result format [8]. Since we add only new reduction rules for SPARQL-like heads and bodies, it is easy to see that each native XQuery is treated in a semantically equivalent way in XSPARQL.

In order to convince the reader that the same holds for native SPARQL queries, we will illustrate our reduction in the following. We restrict ourselves here to a more abstract presentation of our rewriting algorithm, as we implemented it in a prototype.[4]

The main idea behind our implementation is translating XSPARQL queries to corresponding XQueries which possibly use interleaved calls to a SPARQL endpoint. The architecture of our prototype shown in Fig. 10 consists of three main components:

(1) a query rewriter, which turns an XSPARQL query into an XQuery;

(2) a SPARQL endpoint, for querying RDF from within the rewritten XQuery; and

(3) an XQuery engine for computing the result document.

[4]http://xsparql.deri.org/

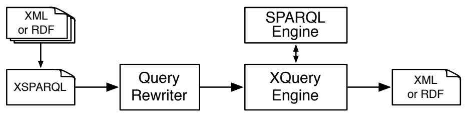


Figure 10: XSPARQL architecture

```
 1 import module namespace _xsparql = "http://xsparql.deri.org/xsparql.xquery"
 2  at "http://xsparql.deri.org/xsparql.xquery";
 3 declare vc = "http://www.w3.org/2001/vcard-rdf/3.0#";
 4 declare foaf = "http://xmlns.com/foaf/0.1/";
 5 declare _sparql_result = "http://www.w3.org/2005/sparql-results#";
 6 declare variable $_NS1 := "prefix vc: <...vcard-rdf/3.0#> ";
 7 declare variable $_NS2 := "prefix foaf: <...foaf/0.1/> ";
 8 fn:concat("@",$_NS1,".","@",$_NS2,"."),
 9 let $_aux1 := fn:concat(
10  "http://localhost:2020/sparql?query=", fn:encode-for-uri(fn:concat($_NS1, $_NS2,
11  "select $P $N $F from <vc.rdf> where {$P vc:Given $N. $P vc:Family $F.}")))
12 for $_aux_result1 at $_aux_result1_pos in doc($_aux1)//_sparql_result:result
13  let $_P_Node := $_aux_result1/_sparql_result:binding[@name="P"]
14  let $P := data($_P_Node/*) let $_P_NodeType := name($_P_Node/*)
15  let $_P_RDFTerm := _xsparql:_rdf_term($_P_NodeType,$P)
16  let $_N_Node := $_aux_result1/_sparql_result:binding[@name="N"]
17  let $N := data($_N_Node/*) let $_N_NodeType := name($_N_Node/*)
18  let $_N_RDFTerm := _xsparql:_rdf_term($_N_NodeType,$N)
19  let $_F_Node := $_aux_result1/_sparql_result:binding[@name="F"]
20  let $F := data($_F_Node/*) let $_F_NodeType := name($_F_Node/*)
21  let $_F_RDFTerm := _xsparql:_rdf_term($_F_NodeType,$F)
22  let $_validSubject1 := fn:concat("_:b", "_", data($_aux_result1_Pos))
23  let $_validObject2 := fn:concat( '"',   fn:concat($N   , " "   , $F   )   ,  '"')
24  return    if ( _xsparql:_validSubject( "",  $_validSubject1  ) ) then (
25    if ( _xsparql:_validObject( "",  $_validObject2  ) ) then (
26      fn:concat( $_validSubject1,  " foaf:name ", $_validObject2, " .&#xA;" )
27    ) else ""    ) else ""
```

Figure 11: XQuery encoding of Example 8(b)

The rewriter algorithm (Fig. 12) takes as input a full XSPARQL *QueryBody* [9] q (i.e., a sequence of **FLWOR'** expressions), a set of bound variables b and a set of position variables p, which we explain below. For a **FL** (or **F'**, resp.) clause s, we denote by $\mathrm{vars}(s)$ the list of all newly declared variables (or the *varlist*, resp.) of s. For the sake of brevity, we only sketch the core rewriting function *rewrite*() here; additional machinery handling the prolog including function, variable, module, and namespace declarations is needed in the full implementation. The rewriting is initiated by invoking *rewrite*$(q, \emptyset, \emptyset)$ with empty bound and position variables, whose result is an XQuery. Fig. 11 shows the output of our translation for the **`construct`** query in Fig. 8(b) which illustrates both the lowering and lifting parts.[5] Let us explain the algorithm using this sample output.

After generating the prolog (lines 1–9 of the output), the rewriting of the Query-Body is performed recursively following the syntax of XSPARQL. During the traversal of the nested **FLWOR'** expressions, SPARQL-like heads or bodies will be replaced by XQuery expressions, which handle our two tasks. The lowering part is processed first:

Lowering The lowering part of XSPARQL, i.e., SPARQL-like **F'DWM** blocks, is "en-

[5]We provide an online interface where other example queries can be found and tested along with a downloadable version of our prototype at `http://www.polleres.net/xsparql/`.

coded" in XQuery with interleaved calls to an external SPARQL endpoint. To this end, we translate **F'DWM** blocks into equivalent XQuery **FLWO** expressions which retrieve SPARQL result XML documents [1] from a SPARQL engine; i.e., we "push" each **F'DWM** body to the SPARQL side, by translating it to a native **`select`** query string. The auxiliary function *sparql*() in line 6 of our rewriter provides this functionality, transforming the **`where`** $\{pattern\}$ part of **F'DWM** clauses to XQuery expressions which have all bound variables in $\mathrm{vars}(pattern)$ replaced by the values of the variables; "free" XSPARQL variables serve as binding variables for the SPARQL query result. The outcome of the *sparql*() function is a list of expressions, which is concatenated and URI-encoded using XQuery's XPath functions, and wrapped into a URI with `http` scheme pointing to the SPARQL query service (lines 10–12 of the output), cf. [8]. Then we create a new XQuery **`for`** loop over variable `$aux_result` to iterate over the query answers extracted from the SPARQL XML result returned by the SPARQL query processor (line 13). For each variable $\$x_i \in \mathrm{vars}(s)$ (i.e., in the (**F'**) **`for`** clause of the original **F'DWM** body), new auxiliary variables are defined in separate **`let`**-expressions extracting its node, content, type (i.e., literal, uri, or blank), and RDF-Term ($\$x_i$`_Node`, $\$x_i$, $\$x_i$`_NodeType`, and $\$x_i$`_RDFTerm`, resp.) by appropriate XPath expressions (lines 14–22 of Fig. 11); the *auxvars*() helper in line 6 of the rewriter algorithm (Fig. 12) is responsible for this.

Lifting For the lifting part, i.e., SPARQL-like **`construct`**s in the **R** part, the transformation process is straightforward. Before we rewrite the QueryBody q, we process the prolog (**P**) of the XSPARQL query and output every namespace declaration as Turtle string literals "`@prefix ns: <URI>.`" (line 10 of the output). Then, the rewriter algorithm (Fig. 12) is called on q and recursively decorates every **`for`** \$ *Var* expression by fresh position variables (line 13 of our example output); ultimately, **`construct`** templates are rewritten to an assembled string of the pattern's constituents, filling in variable bindings and evaluated subexpressions (lines 23–24 of the output).

Blank nodes in **`construct`**s need special care, since, according to SPARQL's semantics, these must create new blank node identifiers for each solution binding. This is solved by "adorning" each blank node identifier in the **`construct`** part with the above-mentioned position variables from any enclosing **`for`**-loops, thus creating a new, unique blank node identifier in each loop (line 23 in the output). The auxiliary function *rewrite-template*() in line 8 of the algorithm provides this functionality by simply adding the list of all position variable p as expressions to each blank node string; if there are nested expressions in the supplied **`construct`** $\{template\}$, it will return a sequence of nested **FLWOR**s with each having *rewrite*() applied on these expressions with the in-scope bound and position variables.

Expressions involving **`construct`**s create Turtle output. Generating RDF/XML output from this Turtle is optionally done as a simple postprocessing step supported by using standard RDF processing tools.

4.4 Correspondence between XSPARQL and XQuery

As we have seen above, XSPARQL syntactically subsumes both XQuery and SPARQL. Concerning semantics, XSPARQL equally builds on top of its constituent languages. We have extended the formal semantics of XQuery from [9] by additional reduction rules which reduce each XSPARQL query to XQuery expressions which operate on results of SPARQL queries in the SPARQL's XML result format [8].

Algorithm 1: *rewrite*(q, b, p) : Rewrite XSPARQL q to an XQuery

```
Input: XSPARQL query $q$, set of bounded variables $b$, set of position variables $p$
Result: XQuery
if $q$ is of form $s_1$ , ... , $s_k$ then
    return rewrite$(s_1, b, p)$ , ... , rewrite$(s_k, b, p)$
else if $q$ is of form for $\$x_1$ in $XPathExpr_1$, ... , $\$x_k$ in $XPathExpr_k$ $s_1$ then
    return for $\$x_1$ at $\$\_x_1\_pos$ in $XPathExpr_1$, ... , $\$x_k$ at $\$\_x_k\_pos$ in $XPathExpr_k$
    rewrite$(s_1, b, p \cup \{\$\_x_1\_pos, \ldots, \$\_x_k\_pos\})$
else if $q$ is of form for $\$x_1 \cdots \$x_n$ from D where { pattern } M $s_1$ then
    return let $\$\_aux\_query$ := sparql(D, $\{\$x_1, \ldots, \$x_n\}$, pattern, M, $b$ ) for $\$\_aux\_result$ in doc ( $\$\_aux\_query$
    )//sparql:result auxvars$(\{\$x_1, \ldots, \$x_n\})$ rewrite$(s_1, b \cup \mathrm{vars}(q), p)$
else if $q$ is of form construct {template} then
    return return ( rewrite-template$(template, b, p)$ )
else
    split $q$ into its subexpressions $s_1, \ldots, s_n$
    for $j := 1, \ldots, n$ do $b_j = b \cup \bigcup_{1 \leq i \leq j-1} \mathrm{vars}(s_i)$
    if $n > 1$ then return $q\,[s_1/rewrite(s_1, b_1, p), \ldots, s_n/rewrite(s_n, b_n, p)]$
    else return $q$
end
```

Figure 12: Algorithm to Rewrite XSPARQL q to an XQuery

Since we add only new reduction rules for SPARQL-like heads and bodies, it is easy to see that each native XQuery is treated in a semantically equivalent way in XSPARQL.

Lemma 28 *Let q be an XSPARQL query. Then, the result of applying q to the algorithm in Fig. 12 is an XQuery, i.e., rewrite$(q, \emptyset, \emptyset)$ is an XQuery.*

Proof. This can be shown straightforwardly by structural induction of q.

Induction base: If q consists of a single XQuery expression i.e. the expression does not use any of the changed grammar rules from Section 4.2, the result of applying *rewrite* is also an XQuery.

If q is composed of several XSPARQL expressions $s_1, \ldots, s_n$ and s_i is an XQuery (from the induction base), s_{i+1} can be of the following cases (presented in Sections 4.2.1 and 4.2.2):

1. If s_{i+1} is in the form of a **SparqlForClause**, the mapping rule
$$\left[\!\!\left[\begin{array}{l} \texttt{for}\ \$\,VarName_1 \cdots \$\,VarName_n\ DatasetClause\ \texttt{where} \\ GroupGraphPattern\ SolutionModifier\ ReturnClause \end{array} \right]\!\!\right]_{Expr'}$$
will convert the XSPARQL query into XQuery;

2. If s_{i+1} is in the form of a **ReturnClause**, by the mapping rule $[\![\texttt{construct}\ ConstructTemplate']\!]_{Expr'}$ it will be converted into XQuery.

□

Proposition 29 *XSPARQL is a conservative extension of XQuery.*

Proof.[Sketch] From Lemma 28, we know that the output, given an XSPARQL query falling in the XQuery fragment is again an XQuery. Note that, however, even this fragment, our additional rewriting rules do change the original query in some cases. More concretely, what happens is that by our "decoration" rule from p.175 each position-variable free **for** loop (i.e., that does not have an **at** clause) is decorated with a new position variable. As these new position variables begin with an underscore they cannot occur in the original query, so this rewriting does not interfere with the semantics of the original query. The only rewriting rules which use the newly created position variables are those for rewriting blanknodes in **construct** parts, i.e., the $[\cdot]_{BNodeSubst}$ rule. However, this rule only applies to XSPARQL queries which fall outside the native XQuery fragment, obviously. $\square$

In order to convince the reader that a similar correspondence holds for native SPARQL queries, let us now sketch the proof showing the equivalence of XSPARQL's semantics and the evaluation of rewritten SPARQL queries into native XQuery. Intuitively, we "inherit" the SPARQL semantics from the *fs*:*sparql* "oracle."

Let Ω denote a solution sequence of a an abstract SPARQL query $q = (E, DS, R)$ where E is a SPARQL algebra expression, DS is an RDF Dataset and R is a set of variables called the query form (cf. [22]). Then, by $SPARQLResult(\Omega)$ we denote the SPARQL result XML format representation of Ω.

We are now ready to state some important properties about our transformations. The following proposition states that any SPARQL **select** query can be equivalently viewed as an XSPARQL **F'DWMR** query.

Proposition 30 *Let* $q = (E_{\mathbf{WM}}, DS, \$x_1, \ldots, \$x_n)$ *be a SPARQL query of the form* **select** $\$x_1, \ldots, \$x_n$ ***DWM****, where we denote by* DS *the RDF dataset (cf. [22]) corresponding to the* $DataSetClause$ *(**D**), by* G *the respective default graph of* DS*, and by* $E_{\mathbf{WM}}$ *the SPARQL algebra expression corresponding to **WM** and* P *be the pattern defined in the* **where** *part (**W**). If* $eval(DS(G), q) = \Omega_1$*, and*

$$statEnv;\ dynEnv \vdash \textbf{\textit{for}}\ \$x_1 \cdots \$x_n \textbf{\textit{from}}\ D(G)\ \textbf{\textit{where}}\ P\ \textbf{\textit{return}}\ (\$x_1, \ldots, \$x_n) \Rightarrow \Omega_2.$$

Then, $\Omega_1 \equiv \Omega_2$ *modulo representation.*[6]

Proof.[Sketch] By the rule

$$[\textbf{for}\ \$x_1 \cdots \$x_n\ \textbf{from}\ D(G)\ \textbf{where}\ P\ \textbf{return}\ (\$x_1, \ldots, \$x_n)]_{Expr'}$$
$$==$$
$$\left[\!\!\left[\textbf{let}\ \$\_aux\_queryresult\ := [\cdot]_{SparqlQuery} \cdots \textbf{for} \cdots [\cdot]_{SparqlResult} \cdots \textbf{return}\ (\$x_1, \ldots, \$x_n)\right]\!\!\right]_{Expr'}$$

$[\cdot]_{SparqlQuery}$ builds q as string without replacing any variable, since all variables in $\mathrm{vars}(P)$ are free. Then, the resulting string is applied to *fs*:*sparql*, which – since q was unchanged – by definition returns exactly $SPARQLResult(\Omega_1)$, and thus the return part **return** $(\$x_1, \ldots, \$x_n)$ which extracts Ω_2 is obviously just a representational variant of Ω_1.

$\square$

By similar arguments, we can see that SPARQL's **construct** queries are treated semantically equivalent in XSPARQL and in SPARQL. The idea here is that the rewriting rules **construct**s from Section 4.2.2 extract exactly the triples from the solution sequence from the body defined as defined in the SPARQL semantics [22].

[6]Here, by equivalence ($\equiv$) modulo representation we mean that both Ω_1 and Ω_2 represent the same sequences of (partial) variable bindings.

5 Related Works

Albeit both XML and RDF are nearly a decade old, there has been no serious effort on developing a language for convenient transformations between the two data models. There are, however, a number of apparently abandoned projects that aim at making it easier to transform RDF data using XSLT. *RDF Twig* [23] suggests XSLT extension functions that provide various useful views on the "sub-trees" of an RDF graph. The main idea of RDF Twig is that while RDF/XML is hard to navigate using XPath, a subtree of an RDF graph can be serialized into a more useful form of RDF/XML. *TreeHugger*[7] makes it possible to navigate the graph structure of RDF both in XSLT and XQuery using XPath-like expressions, abstracting from the actual RDF/XML structure. *rdf2r3x*[8] uses an RDF processor and XSLT to transform RDF data into a predictable form of RDF/XML also catering for RSS. Carroll and Stickler take the approach of simplifying RDF/XML one step further, putting RDF into a simple *TriX* [6] format, using XSLT as an extensibility mechanism to provide human-friendly macros for this syntax.

These approaches rely on non-standard extensions or tools, providing implementations in some particular programming language, tied to specific versions of XPath or XSLT processors. In contrast, *RDFXSLT*[9] provides an XSLT preprocessing stylesheet and a set of helper functions, similar to RDF Twig and TreeHugger, yet implemented in pure XSLT 2.0, readily available for many platforms.

All these proposals focus on XPath or XSLT, by adding RDF-friendly extensions, or preprocessing the RDF data to ease the access with stock XPath expressions. It seems that XQuery and SPARQL were disregarded previously because XQuery was not standardized until 2007 and SPARQL – which we suggest to select relevant parts of RDF data instead of XPath – has only very recently received W3C's recommendation stamp.

As for the use of SPARQL, Droop et al. [10] suggest, orthogonal to our approach, to compile XPath queries into SPARQL. Similarly, encoding SPARQL completely into XSLT or XQuery [15] seems to be an interesting idea that would enable to compile down XSPARQL to pure XQuery without the use of a separate SPARQL engine. However, scalability results in [15] so far do not yet suggest that such an approach would scale better than the interleaved approach we took in our current implementation.

Finally, related to our discussion in Section 2, the SPARQL Annotations for WSDL (SPDL) project (`http://www.w3.org/2005/11/SPDL/`) suggests a direct integration of SPARQL queries into XML Schema, but is still work in progress. We expect SPDL to be subsumed by SAWSDL, with XSPARQL as the language of choice for lifting and lowering schema mappings.

6 Conclusion and Future Plans

We have elaborated on use cases for lifting and lowering, i.e., mapping back and forth between XML and RDF, in the contexts of GRDDL and SAWSDL. As we have seen, XSLT turned out to provide only partially satisfactory solutions for this task. XQuery

[7]`http://rdfweb.org/people/damian/treehugger/index.html`
[8]`http://wasab.dk/morten/blog/archives/2004/05/30/transforming-rdfxml-with-xslt`
[9]`http://www.wsmo.org/TR/d24/d24.2/v0.1/20070412/rdfxslt.html`

and SPARQL, each in its own world, provide solutions for the problems we encountered, and we presented XSPARQL as a natural combination of the two as a proper solution for the lifting and lowering tasks. Moreover, we have seen that XSPARQL offers more than a handy tool for transformations between XML and RDF. Indeed, by accessing the full library of XPath/XQuery functions, XSPARQL opens up extensions such as value-generating built-ins or even aggregates in the construct part, which have been pointed out missing in SPARQL earlier [21].

As we have seen, XSPARQL is a conservative extension of both of its constituent languages, SPARQL and XQuery. The semantics of XSPARQL was defined as an extension of XQuery's formal semantics adding a few normalization mapping rules. We provide an implementation of this transformation which is based on reducing XSPARQL queries to XQuery with interleaved calls to a SPARQL engine via the SPARQL protocol. There are good reasons to abstract away from RDF/XML and rely on native SPARQL engines in our implementation. Although one could try to compile SPARQL entirely into an XQuery that caters for all different RDF/XML representations, that would not solve the use which we expect most common in the nearer future: many online RDF sources will most likely not be accessible as RDF/XML files, but rather via RDF stores that provide a standard SPARQL interface.

Our resulting XSPARQL preprocessor can be used with any available XQuery and SPARQL implementation, and is available for user evaluation along with all examples of this paper at `http://xsparql.deri.org/`.

As mentioned briefly in the introduction, simple reasoning – which we have not yet incorporated – would significantly improve queries involving RDF data. SPARQL engines that provide (partial) RDFS support could immediately address this point and be plugged into our implementation. But we plan to go a step further: integrating XSPARQL with Semantic Web Pipes [18] or other SPARQL extensions such as SPARQL++ [21] shall allow more complex intermediate RDF processing than RDFS materialization.

We also plan to apply our results for retrieving metadata from context-aware services and for Semantic Web service communication, respectively, in the EU projects inContext (`http://www.in-context.eu/`) and TripCom (`http://www.tripcom.org/`).

References

[1] Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format, November 2007. W3C Proposed Recommendation, available at `http://www.w3.org/TR/2007/PR-rdf-sparql-XMLres-20071112/`.

[2] Dave Beckett and Brian McBride (eds.). RDF/XML Syntax Specification (Revised). Technical report, W3C, February 2004. W3C Recommendation.

[3] David Beckett. Turtle - Terse RDF Triple Language, November 2007. Available at `http://www.dajobe.org/2004/01/turtle/`.

[4] Tim Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. Internet Engineering Task Force RFC 3986, Internet Society (ISOC), January 2005. Published online in January 2005 at `http://tools.ietf.org/html/rfc3986`.

[5] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. W3C recommendation, World Wide Web Consortium, August 2006. Published online in August 2006 at http://www.w3.org/TR/REC-xml-names.

[6] Jeremy Carroll and Patrick Stickler. TriX: RDF Triples in XML. Available at http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html.

[7] Don Chamberlin, Jonathan Robie, Scott Boag, Mary F. Fernández, Jérôme Siméon, and Daniela Florescu. XQuery 1.0: An XML Query Language. W3C recommendation, W3C, January 2007. W3C Recommendation, available at http://www.w3.org/TR/xquery/.

[8] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. SPARQL Protocol for RDF, November 2007. W3C Proposed Recommendation, available at http://www.w3.org/TR/2007/PR-rdf-sparql-protocol-20071112/.

[9] Denise Draper, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics. W3c recommendation, W3C, January 2007. W3C Recommendation, available at http://www.w3.org/TR/xquery-semantics/.

[10] Matthias Droop, Markus Flarer, Jinghua Groppe, Sven Groppe, Volker Linnemann, Jakob Pinggera, Florian Santner, Michael Schier, Felix Schöpf, Hannes Staffler, and Stefan Zugal. Translating xpath queries into sparql queries. In *6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, 2007.

[11] Dan Connolly (ed.). Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C recommendation, W3C, September 2007.

[12] Michael Kay (ed.). XSL Transformations (XSLT) Version 2.0 , January 2007. W3C Recommendation, available at http://www.w3.org/TR/xslt20.

[13] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007.

[14] Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C, August 2007. Available at http://www.w3.org/TR/sawsdl/.

[15] Sven Groppe, Jinghua Groppe, Volker Linneman, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke. Embedding SPARQL into XQuery/XSLT. In *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC2008)*, March 2008. To appear.

[16] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[17] Ashok Malhotra, Jim Melton, and Norman Walsh (eds.). XQuery 1.0 and XPath 2.0 Functions and Operators , January 2007. W3C Recommendation, available at http://www.w3.org/TR/xpath-functions/.

[18] Christian Morbidoni, Axel Polleres, Giovanni Tummarello, and Danh Le Phuoc. Semantic web pipes. Technical Report DERI-TR-2007-11-07, DERI Galway, 11 2007.

[19] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International Semantic Web Conference (ISWC 2006)*, pages 30–43, 2006.

[20] Axel Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th World Wide Web Conference (WWW2007)*, Banff, Canada, May 2007.

[21] Axel Polleres, François Scharffe, and Roman Schindlauer. SPARQL++ for mapping between RDF vocabularies. In *6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 878–896, Vilamoura, Algarve, Portugal, November 2007. Springer.

[22] Eric Prud'hommeaux and Andy Seaborne (eds.). SPARQL Query Language for RDF, January 2008. W3C Recommendation, available at `http://www.w3.org/TR/rdf-sparql-query/`.

[23] Norman Walsh. RDF Twig: Accessing RDF Graphs in XSLT. Presented at Extreme Markup Languages (XML) 2003, Montreal, Canada. Available at `http://rdftwig.sourceforge.net/`.