A Higher-Level View of Ontological Modeling

Rule-Based Approaches for Data Transformation, Modeling, and Maintenance

Dissertation for the degree of Philosophiae Doctor (PhD)

Daniel P. Lupp

University of Oslo
April 2019

© Daniel P. Lupp, 2019

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2108

ISSN 1501-7710

All rights reserved. No part of this publication may be
reproduced or transmitted, in any form or by any means, without permission.

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

To my friends and family

“We’ve got to have rules [...] After all, we’re not savages.”
— William Golding

Contents

Acknowledgments

These last few years have been by far the most taxing years of my life, both on a personal and professional level. This thesis would not have been possible without the substantial support from my colleagues, friends, and family to whom I would like to extend my gratitude.

First I would like to thank all of my advisors Evgenij Thorstensen, Henrik Forssell, and Arild Waaler for their never-ending support and patience as well as understanding in times when things did not go so well. I wish to thank my primary advisor Evgenij for his critical questions, guidance, and advice as well as always helping me look at things from different perspectives. I thank Henrik for his down-to-earth yet optimistic outlook, his uncanny ability to discover counter-examples, and for always helping me see the connections between our work and my previous studies in algebra. I would also like to thank Arild for his constant enthusiasm and support, for always helping me see the big picture, as well as sparking my interest in nonmonotonic reasoning.

I would like to thank all of my co-authors for the opportunity to collaborate with them. In particular, I would like to thank Martin Skjæveland for inviting me to join the OTTR team. More generally I wish to extend my gratitude to all of LogID/ASR/SIRIUS for the interesting discussions and valuable input. Thank you to Mantas Šimkus for hosting my 3-month research visit in Vienna and for the resulting dialogue. Furthermore, a special thanks to my friends, office mates, and collective "Norwegian mother" Vidar, Leif Harald, and Andreas for making office and every-day life so enjoyable. Without their patience, my Norwegian would surely never quite have gotten to where it is today. I would also like to the thank the person(s) at ifi responsible for the grand piano on the fifth floor as this served as an excellent form of stress relief during long nights.

Finally, I wish to thank all of my family and friends for their constant support and kindness despite my failings at describing what it is that I do.

Part I
Summary

Chapter 1
Introduction

The processing of data is becoming an increasingly important and difficult task. Data is stored using a wide range of different formats and largely requires specialized software in order to be accessed. The complex information needs of an end-user require in-depth knowledge of how the data is stored in order to find the desired data. Thus, in addition to their domain of interest, users must either learn the ins-and-outs of their data stores or be reliant on dedicated IT experts in order to access the information they need.

To overcome these challenges, data from multiple sources can be integrated under a single, global schema. In this approach data is extracted from the relevant data sources and transferred into a database under the global schema. This approach is known as *extract, transform, load* (ETL) and is commonly used in data warehousing. The global schema is highly specific to the original data sources and, in order to ensure efficiency, is highly optimized towards these sources and the expected information needs. This is by no means a novel approach: since the 1980's businesses have employed these techniques in order to transfer data from multiple databases into a data warehouse. To this day, such solutions greatly benefit industrial decision making and reporting as it facilitates the aggregating and processing of all relevant data in one place.

In more recent years, however, the sheer amount and heterogeneity of data has changed substantially. In a press article in 2013, SINTEF researchers estimated that at the time of writing over 90% of all data ever gathered was generated after 2011.[1] In addition to the exponential growth in volume, data is being gathered from a wide variety of both structured and unstructured sources, e.g., sensors and measurements, social media, as well as structured/unstructured natural language documents. As a result, the information needs are evolving as well, where companies attempt to incorporate as much of this new information as possible into their decision making

[1] `https://www.sintef.no/en/latest-news/big-data-for-better-or-worse/`

processes.

As the nature of industry-relevant data becomes more and more varied, the ETL approach of gathering and transferring all of it into a global schema becomes, in general, an expensive and practically infeasible task. Any newly collected data would need to be translated and transferred, resulting in enormous runtime costs. Furthermore, a global schema would need to be highly complex in order to accommodate the large variety of data gathered, ranging from legacy company databases to streaming data and PDF documents. Maintenance tasks become infeasible to perform, as every time a new source is added the global schema must be adjusted. As a result, all existing data must then be re-processed and transferred in order to comply with the updated schema. This makes maintenance a very expensive task, in addition to making it extremely difficult for end-users to access the data they need without an IT expert's help.

Rather than defining a global schema that represents specific data sources, recent research on data integration has focused on using an external abstract model in place of a global schema [30]. This abstract model, called an *ontology*, ideally represents an end-user's domain of interest and should be independent of how the data is stored. In *ontology-based data access*, the data is not actually transferred into a new database, but rather remains in its original sources. Instead, the ontology is connected to data sources with the help of mappings which translate between queries over the ontology to queries over the data sources. As such, the mappings handle the intricacies of each data source as opposed to the ontology; end-users are ideally not burdened with technical details on how the data is stored or translated. Thus, they can phrase queries in a language closer to their understanding of the domain as opposed to a data-specific querying language. Since no data is actually translated, any change to the ontology or mappings does not result in all data needing to be re-evaluated.

Adding this extra layer of abstraction over the data has a number of benefits for the end-user. The ontology should be designed to closely represent the domain they know and understand, simplifying query formulation tasks. Furthermore, the user queries become data independent, resulting in a solution that scales well with additional data sources—provided the mappings are being actively maintained.

Of course, this approach comes with its own difficulties. The construction of a high-quality ontology is by no means a simple task. In fact, what constitutes a high-quality ontology is still hotly debated and an active field of research. Nevertheless, it should reflect the end-user's understanding of the domain while keeping the information requirements in mind, hence requiring close collaboration between the intended users and ontology engineers. "What will the ontology be used for?" is a crucial question to be asked when designing an ontology, as it greatly affects the decisions on how to model certain relationships and patterns. In addition, the ontology and mappings should be easily maintainable, allowing for automated processes whenever possible. This greatly reduces maintenance overhead and eliminates potential sources of human error.

The work presented in this thesis is aimed at addressing issues that arise during the creation and maintenance of ontologies and mappings. More specifically, the goals are to

1. extend mappings to robustly handle exceptions as well as incomplete data, thus simplifying mapping maintenance by eliminating the need to explicitly list all exceptions;

2. facilitate the definition and reuse of recurring patterns within an ontology in order to ensure a uniform modeling approach, thus simplifying ontology creation and maintenance tasks

by separating the modeling and populating of an ontology;

3. provide mechanisms for the detection and removal of redundancies within an ontology and the patterns it contains, thereby providing semi-automatic tools for ontology maintenance and pattern discovery;

4. define a formalism that supports the encoding of design choices regarding dependencies between ontology patterns, thus automating previous manual maintenance and creation tasks.

To this end, we define suitable rule-based formalisms over (1) the data and (2) over the ontology. The former provides a new mapping framework that simplifies mapping maintenance tasks by facilitating robust handling of exceptions and incomplete data (see [P1]). For the latter, templates for ontology specification are introduced as a means to robustly define and instantiate recurring patterns within an ontology, while supporting semi-automatic redundancy detection [P2, P3, P4]. Extending ontology templates are rules called generators, which allow for the succinct description of dependencies between pattern instantiations in an ontology [P6].

In the remainder of this chapter, we briefly introduce important notions and the context of this thesis, in particular ontology-based data integration and rule-based formalisms for knowledge representation. In Chapter 2, we take a closer look at the scientific contributions of the papers contained in Part II.

1.1 Ontologies and Data Integration

Ontologies

In general, an *ontology* is a machine-readable model designed to faithfully represent knowledge of a domain. Such a model provides a way of exploring the domain knowledge as well as inferring implicit information from data with the help of automated reasoning. Several languages have been proposed for expressing ontologies. However, since the early to mid 2000's the emergence of semantic technologies and standardization efforts such as the *Web Ontology Language* (OWL) [5] have ensured that *description logics* (DL's) [4] have become the defacto standard for ontologies in practice. These are a family of logical formalisms designed for knowledge representation and reasoning in the fields of AI and the Semantic Web. Description logics model how individuals from an application domain are grouped into classes (called *concepts*) and related to one another with binary relations (called *roles*). Complex concepts and roles can be built inductively according to the syntax allowed by a specific description logic. An ontology then relates these concepts, roles, and individuals to one another through inclusion axioms $\mathsf{B} \sqsubseteq \mathsf{C}$ (resp. $\mathsf{R} \sqsubseteq \mathsf{S}$) that state that the concept B is a subclass of C (resp. the role R is a subrole of the S) as well as assertions $\mathsf{C}(\mathsf{a})$ (resp. $\mathsf{R}(\mathsf{a}, \mathsf{b})$) that state that the individual a is member of the concept C (resp. a is R-related to b). These formalisms have been defined with syntactic restrictions in mind: different description logics restrict which complex concepts and axioms may be constructed. In this manner, different DL's are designed with specific tasks in mind, e.g., low complexity of reasoning. For example, in order to retain low complexity of entailment the DL-Lite family of description logics [6] does not permit axioms of the form

$\mathsf{A} \sqsubseteq \mathsf{B} \sqcup \mathsf{C}$, i.e., it is not permissible to say that a concept is a subclass of a union of other concepts. Indeed, DL-Lite is an important class; in the following section we discuss how the complexity of entailment is important in the context of ontology-based data integration, as it is tightly linked to query rewritability. For details on this as well as an an overview of the DL's associated with the OWL profiles, see [20].

We shall use description logic notation for notational convenience throughout this thesis when discussing example ontologies (for more details see [4]).

Example 1. The statement "every boss is someone's superior" can be modeled by the axiom $\mathsf{Boss} \sqsubseteq \exists \mathsf{hasSup}^{-}.\top$ where hasSup^{-} refers to the *inverse role* of hasSup. This axiom is equivalent to the first-order formula $\forall \mathsf{x}.(\mathsf{Boss}(\mathsf{x}) \to \exists \mathsf{y}.\mathsf{hasSup}(\mathsf{y}, \mathsf{x}))$.

Creating and maintaining a high-quality, large-scale ontology typically involves

1. identifying what concepts and relationships from the domain need to be modeled;
2. deciding on an appropriate way to model these according to the information requirements given by users; and
3. maintaining the ontology, e.g., ensuring uniformity in modeling, removing unwanted redundancies.

Steps 1 and 2 above are to a large degree a creative process: Step 1 requires active communication between ontology engineers and experts from the domain to be modeled, a task which cannot be fully automated. Step 2 involves finding a best-practice way of modeling the desired relationships. This is by no means a straightforward task and has evolved into its own area of research called *ontology design patterns* (ODP's). Inspired by software design patterns, ODP's strive to provide modeling references for often-occurring and important patterns within ontologies [15]. However, one of the major challenges faced by the ODP community is a lack of tool-sets geared towards reuse and maintenance of existing ODP's [15, 16]. This is not due to a lack of tools and approaches for ontology engineering, however. Protégé [29] is widely regarded as the gold standard when it comes to current ontology editors. Indeed, it contains much needed functionality and due to its extensible environment supports a variety of plugins. Protégé plugins such as XDP [14] attempt to alleviate some of the issues faced when using recurring patterns, e.g., by supporting the finding and importing of existing ODP's. However, it does not address some key issues when it comes to ontology and pattern maintenance:

Modular view of ontology The ontology when viewed is still an OWL ontology. As such, it is difficult to determine which axioms constitute an instantiation of a specific pattern.

Change in pattern If a pattern that has been instantiated is changed, each instantiation in the ontology must be manually updated to incorporate these changes.

Change in ontology If one wishes to remove a pattern instance from the ontology, one must manually remove each axiom belonging to that instance.

Other frameworks attempt at addressing these issues by defining new languages with which to specify ontologies. Tawny-OWL [25] and the Ontology Pre-Processing Language (OPPL) [7] are such formalisms for specifying and manipulating ontologies. Tawny-OWL provides a fully programmatic environment based on the programming language Clojure for creating ontologies while including strong support for ontology design patterns. OPPL was designed specifically for capturing patterns and regularities in ontologies, allowing the instantiation of patterns as well as supporting features such as removing axioms. While these formalisms certainly are powerful and there are many advantages to using them, they also come with certain drawbacks. For instance, both require users to learn new syntax for manipulating ontologies. While Tawny-OWL does this by design (it is intended to be more akin to programming than other previously existing ontology editing tools), it is nonetheless a hurdle for users who are not familiar with that specific way of thinking. Furthermore, the semantics of the formalisms are fairly complicated, or even unclear: OPPL supports a removal action, which unto itself already results in a necessarily non-monotonic semantics. It is likely that this was not a key factor when designing these formalisms (rather focusing on having the frameworks support the desired functionality), however having a clear semantics with decidable (or even tractable) entailment provides much utility for maintenance tasks.

In [P2, P3, P4], we introduce the OTTR framework, a templating language for the definition and instantiation of ontology patterns. It is specifically designed to provide a modular view of the ontology while supporting robust mechanisms for changes in both the pattern and the ontology. It can be easily adopted without the need to learn an entirely new syntax and has clear semantics, which permits the semi-automation of certain maintenance tasks such as detecting and removing redundancies. These contributions are summarized in Section 2.2.

Ontology-based Data Integration

Ontology-based data integration (OBDI) is a specific form of data integration where an ontology plays the role of the global schema. Such a format allows for a variety of uses, such as data transformation (where the ontology plays the role of an intermediary, translating theory between different database schemas) and data access (where the ontology is used as a querying language over disparate sources). The papers in Part II of this thesis are primarily focused on the data access setting, though the results are transferable to any OBDI context.

Thus, ontology-based data access (OBDA) [30] utilizes the semantic layer consisting of an ontology and a set of mappings on top of the data in a database. With the help of mappings, queries over the ontology are translated into a query over the database language, such as SQL, which can then be run on the source data. In order to ensure that all relevant data is retrieved, the query translation consists of two stages: Firstly, the ontology query is rewritten to an equivalent query which takes ontological knowledge into account. Secondly, the mappings are used to translate this query into the source query language.

Example 2. Consider a database consisting of precisely one table `JOBS_DB` with the two columns `<NAME>` and `<JOB>`. Furthermore, consider the ontology $\{\mathsf{Empl} \sqsubseteq \mathsf{Person}, \mathsf{Boss} \sqsubseteq \mathsf{Person}\}$. In the rewriting process, the query $\mathsf{Person(x)}$ would be rewritten to $\mathsf{Person(x)} \sqcup \mathsf{Empl(x)} \sqcup \mathsf{Boss(x)}$ while in the unfolding step, each of the above disjuncts would be expanded to a database query using the mapping assertions. For example, if there exist two mapping assertions $\mathtt{JOBS\_DB}(\mathsf{x}, \text{“}Accountant\text{”}) \rightarrow \mathsf{Empl(x)}$ and $\mathtt{JOBS\_DB}(\mathsf{x}, \text{“}IT\text{”}) \rightarrow \mathsf{Empl(x)}$, then the disjunct $\mathsf{Empl(x)}$ would be unfolded as $\mathtt{JOBS\_DB}(\mathsf{x}, \text{“}IT\text{”}) \cup \mathtt{JOBS\_DB}(\mathsf{x}, \text{“}Accountant\text{”})$.

In general, it is not possible to employ this technique in order to rewrite an arbitrary ontology query to a source query [6, 20]. Hence, one poses certain criteria on the ontology and queries in order to ensure rewritability. Since the database querying language SQL is very closely tied to first-order logic [1], a common such criterion is that they are *first-order rewritable* (FOL-rewritable); that is, that the ontology rewriting of every permitted query is equivalent to a first-order formula. However, not all description logics have this property. Indeed, any description logic where the entailment problem is NLOGSPACE-hard in data complexity[2] cannot be FOL-rewritable [20].

A popular class of ontology languages for ontology-based data access is the DL-lite family. These description logics have been tailored specifically for FOL-rewritability by restricting ontology axioms in order to ensure low complexity entailment. This makes them ideally suited for OBDA [6].

Example 2 demonstrates some of the current shortcomings of classical OBDA mappings: it is impossible to distinguish between inferred knowledge and knowledge that is explicit in the database. In the above example, in the presence of a mapping assertion $\mathtt{JOBS\_DB}(\mathsf{x},\mathsf{y}) \rightarrow \mathsf{Person}(\mathsf{x})$ the query $\mathsf{Person}(\mathsf{x})$ would have sufficed without any ontology rewriting, since all desired information was contained in one table. However, while some OBDA implementations support manual query pruning [18] this can potentially lead to incomplete query answering, and there is currently no way of formally checking whether it does. As such, the rewriting step can cause a worst-case exponential blow-up in query size [6]. While this blow-up is necessary to ensure complete query answering, it can lead to highly redundant database queries, where the same data is accessed multiple times. Methods for automatic pruning of queries where the mappings and ontology are anaylzed to determine redundant queries are used in practice [32]. However, due to its foundation in first-order logic, the current logical framework underlying OBDA does not support the ability to express which queries *should* be redundant, e.g., that all Persons should be contained in the JOBS_DB table. Extensions to description logics that support the necessary non-monotonic features such as extensional constraints [33] or closed predicates [26] have been proposed. However, due to an increase in data complexity these quickly become practically infeasible.

Another issue with the current approach is how exceptions and incomplete information are dealt with. Currently, one must keep track of exceptions manually by explicitly listing all exceptions to a rule. Thus, if new exceptions must be taken into consideration, all mappings must be manually updated to ensure sound and complete query answering.

In general, mapping design and maintenance tends to be primarily manual work [3]. This can be a very laborious task and work on mapping evolution and repair [21] attempts to mitigate some of these difficulties. A mature method for encoding exceptions without the need to explicitly list all cases promises to reduce this cost even further. In [P1] we introduce a new mapping framework called *mapping programs* designed specifically to address this issue. This is summarized in Section 2.1.

[2]Data complexity refers to the complexity when the size of data may vary but the size of the query and ontology are fixed. If the ontology and query may vary in size one speaks of *combined complexity.* This result tying data complexity and FOL-rewritability relies on the fact that the size of the query and ontology are fixed.

1.2 Logic Programming

Datalog and Answer Set Programming

For many decades, logic programming has played an important role within both the knowledge representation and data integration fields. Since the late 70's and early 80's, a declarative fragment of the logic programming language Prolog known as *Datalog* has become a prominent query and data transformation language as it supports features not expressible in the relational calculus, e.g., recursive queries while maintaining a clear semantics with low complexity reasoning [1].

Datalog programs consist of *rules*, which are Horn clauses of the form

$$H \leftarrow B_1, \dots, B_m,$$

where H and $B_1, \dots, B_m$ are positive literals, i.e., atomic formulas. $\{H\}$ is referred to as the the *head* and $\{B_1, \dots, B_m\}$ as the *body* of the rule. Rules with an empty body are called *facts*, whereas rules with an empty head are called *constraints*.

Since Datalog allows for any positive literal in its rules' bodies, it does not have the same restriction to binary predicates as description logics have. Furthermore, due to the generality of such rules, recursive queries can be formulated. This allows for, e.g., checking the transitive closure in a graph using Datalog rules.

Example 3. Given a graph with a binary edge relation E, the Datalog rule

$$E(x,z) \leftarrow E(x,y), E(y,z)$$

generates the transitive closure.

The intuition behind the semantics of a Datalog rule is that if $B_1, \dots, B_m$ are true then H must be true. More formally, the semantics of Datalog are defined using an iterative fixpoint construction which is guaranteed to terminate for any Datalog program applied to a finite instance [1]. Indeed, this fixpoint coincides with the minimal model (unique up to isomorphism) when considering the Datalog program as a first-order theory.

A natural extension to Datalog is allowing for body literals to be negated. Many approaches exist to introduce negation to Datalog [1]. However we shall focus on one in particular: *negation-as-failure*, where "**not** B" is evaluated to true iff no combination of rules implies B [1]. Intuitively, this corresponds to assuming everything but the provably true to be false. This interpretation of negation follows the *closed-world assumption*, which is a conceptual decision on how to handle incomplete information. Intuitively, the closed-world assumption states that the truth of a statement must be justified; a statement A is considered false in a model of a Datalog program if no combination of rules imply A. This is in contrast to the *open-world assumption*, which states that the truth of a statement need not rely on any justification. The classical semantics of propositional logic, for example, employs the open-world assumption: The implication $A \rightarrow B$ does contain any justification for C being true, but this does not imply that C is false.

Unfortunately, adding negation to rule bodies is not as straightforward as one would like. In particular, we lose model uniqueness. Consider the rules

$$B \leftarrow \textbf{not}\, C.$$
$$C \leftarrow \textbf{not}\, B.$$

This program has two distinct minimal models: one where only B is true and one where only C is true.

How to choose and compute the desired models has been (and still is to some degree) a highly-researched field. Certain cases of Datalog programs with negation, such as semipositive or stratified Datalog, can be given a semantics that distinguish a unique minimal model following the closed-world assumption [1]. When this unique model does not exist, however, things become much more difficult and open to debate.

Answer set programming (ASP) is such an approach to adding negation-as-failure to Datalog with a clear manner of handling multiple models. It is a declarative programming paradigm based on the stable model semantics first defined in [12] as a means of handling negation-as-failure in a clear and straightforward manner. It has become one of the more popular logic programming paradigms, due to, e.g., computational benefits such as guaranteed termination as compared to resolution in Prolog [22].

An *ASP-program* P is then a set of rules of the form $H \leftarrow B_1, \ldots, B_m, \textbf{not}\, C_1, \ldots, \textbf{not}\, C_n.$ with ground atoms H, B_i, and C_j. The semantics of an ASP program P are defined through a construction called the *Gelfond-Lifschitz reduct*. Given an interpretation I of ground atoms, the Gelfond-Lifschitz reduct P^I is a positive program such that I satisfies a rule in P^I if and only if I satisfies the corresponding rule in P. Intuitively, this construction is achieved by removing from P all rules that cannot fire under I and then removing all negative clauses $\textbf{not}\, C$ in the remaining rules (for more details see [12, 22]).

The reduct is a program without any occurrence of negation-as-failure, hence is simply a positive Datalog program. Thus, there exists a unique minimal model of the reduct. Then an interpretation I is called a *stable model* or an *answer set* of P if it is a $\subseteq$-minimal model of P^I, i.e., it is a $\subseteq$-minimal set that is also the minimal model of P^I.

Though the above semantics requires ground atoms, i.e., are essentially propositional, ASP programs might also contains variables or function symbols. In this general case where function symbols are allowed, reasoning becomes undecidable [2]. In the function-free case, the first-order ASP programs are usually first grounded to reduce it to the propositional case. The grounded programs can then either be solved directly [11] or, e.g., translated to an instance of the Boolean satisfiability problem (SAT) before being passed on to efficient SAT solvers [24, 13].

The formalisms defined in [P1] and [P6] are extensions of Datalog with negation. The former introduces mapping programs, an extension of ASP, as a new approach to mapping data to ontologies in order to robustly handle exceptions in data. The latter is an extension of Datalog that allows the succinct description of dependencies between ontology patterns. The contributions of these papers are summarized in Section 2.1 and 2.3, respectively.

1.3 Brief Summary of Contributions

The following is a brief summary of the contributions of the included research papers. The overarching goal of the papers in this thesis is to facilitate robust specification and maintenance of ontologies and mappings within the OBDA (and more generally the OBDI) setting. Intuitively, we wish to be able to provide a higher-level view of an OBDI system to ensure better readability, while simultaneously providing new tools for maintenance tasks. The scientific contributions of the papers included in this thesis can be divided into three groups: (1) rules over the data, (2) ontology templates, and (3) rules over ontologies. In the following we briefly summarize the contributions made to each of these categories. For a more detailed description see Chapter 2 of this thesis.

Rules over data Current OBDA mappings are first-order implications employing the open-world assumption. We propose an alternative rule-based formalism, based on answer set programming (ASP), for the mappings linking the data and ontology. By allowing negation-as-failure under the stable model semantics, this formalism allows for a more robust handling of incomplete data and exceptions as compared to its first-order mapping counterparts. Additionally, it is possible to phrase constraints over the data in the language of the ontology, thus allowing the encoding of domain and system-specific knowledge into the mappings. This is in contrast to the current approach, where such information is implicit and not formally checkable within the system.

Ontology templates We provide a new templating language for DL/OWL ontologies. Recurring patterns can be defined as templates, which in turn can be instantiated within an ontology. The proposed formalism has a formal semantics based on the underlying description logics. This allows for robust, automatic checking of formal properties such as consistency and redundancy. Ontologies can be viewed in their expanded form (i.e., as DL ontologies) or unexpanded form. The latter provides a higher-level view on ontologies, enabling (1) a separation of concerns within ontology design concerning *what* relationships are modeled in an ontology and *how* they are modeled, and (2) improved readability and maintainability of large-scale ontologies.

Rules over ontologies As opposed to using rules to extend the expressivity of an ontology, we rather employ logic programming to simplify the specification and maintenance of an ontology while retaining its expressivity. *Generators* are an extension of Datalog that allow for describing the relationships between recurring patterns within an ontology. On the one hand, this ensures a uniformity in the ontological modeling, inheriting the benefits from ontology templates. On the other hand, it provides a robust and scalable way of ensuring that ontology design choices are consistently implemented.

Chapter 2
Contributions

As mentioned in Chapter 1, the overarching goal of this thesis is to simplify specification and maintenance tasks in ontology-based data integration with the help of rule-based formalisms. The contributions of this thesis can be grouped into three categories: (1) rules over data, (2) ontology templates, and (3) rules over ontologies. In the remainder of this chapter, we explore these contributions in greater detail.

2.1 Rules over Data

Combining rules and ontologies is by no means a new field of research. Indeed, several rule-based formalisms have been proposed specifically for interacting with DL ontologies, such as DL-safe rules [28], MKNF$^+$ knowledge bases [27], and dl-programs [9].

DL-safe rules are a restriction to the Semantic Web Rule Language (SWRL) [17], allowing for axiom-like rules that are not expressible in standard description logics. While SWRL rules are in general undecidable, a dl-safe rule employ a safety condition to ensure decidability: any variable must appear in a non-DL atom in the rule body.[1] MKNF$^+$ knowledge bases were designed specifically to integrate open-world ontology with closed-world rule-based reasoning. Its intention is for rules and ontologies to live side-by-side in the same formalism, To that end, they introduce a rule-language building on the logic of minimal knowledge and negation-as-failure (MKNF) defined by Lifschitz [23]. The result is a very expressive formalism capable of expressing ontology axioms as well as nonmonotonic rules. DL-programs add nonmonotonic reasoning to ontologies by allowing rules to contain queries to an external ontology. These queries provide a limited amount of interaction between the ontology and rules: a query can

[1]Mapping programs, as discussed in the following section, employ a similar technique to ensure decidability.

temporarily extend the ontology with facts generated by the program. Due to this manner of interaction, dl-programs are often considered as "rules on top of ontologies:" though they may query ontologies (and during the querying process incorporate facts gained by the program) a dl-program does not alter an ontology.

Rather than using rules to alter or extend the expressivity of an ontology in an OBDA system, we propose using a rule-based formalism in its mappings. In [P1], we define *mapping programs* as a rule language for ontology-based data integration based on answer set programming. As such, each mapping rule contains a database query in its body and an ontology query in its head. Additionally, rule bodies may contain both positive and negated ontology queries as conditions for the mapping. Intuitively, a mapping rule

$$\texttt{DB\_Query}(\vec{x}), O_1(\vec{y_1}), \textbf{not } O_2(\vec{y_2}) \rightarrow O_3(\vec{x}, \vec{z})$$

where $\vec{y_1}, \vec{y_2} \subseteq \vec{x}$ states that all query answers to `DB_Query` should be mapped to O_3 if the mapping program and ontology entails O_1 and not O_2.

The variables $\vec{z}$ in the head of the rule are considered existentially quantified variables, and hence such programs need some sort of safety condition to ensure decidability in query answering. Thus we employ a method akin to the dl-safety condition: Intuitively, a rule may only fire for "known" individuals. In the OBDI context, this corresponds to rules only being allowed to fire for tuples within the database. For mapping programs this is achieved through the presence of a database query within each mapping rule and the GAV condition on the rule head: that all non-existentially quantified variables in the head of the rule are guarded by `DB_Query`.

Mapping programs are similar to dl-programs. In fact, if one adds support for existential quantification in rule heads and a database query in the rule body, they can be translated into an equivalent dl-program. However, mapping programs are purposely restricted; the full expressivity of dl-programs is not required for the purposes for which mapping programs were intended. These restrictions allow, for example, a natural translation from mapping programs to classical ASP if all ontology queries in the mapping program are conjunctive queries [P1].

By mapping data to ontologies in this manner, OBDA systems benefit in the following ways:

Bridge open and closed world assumptions Being able to handle both open-world and closed-world reasoning is a popular motivation for combining logic programming with negation and ontologies [9, 27]. Similar to [8], our approach treats both the ontology and the database as external sources. With existentially quantified variables in the head and negation-as-failure, mapping programs can seamlessly translate from the closed-world database to the open-world ontology without the need for the ontology formalism to be changed. By adding these features to the mappings as opposed to the ontology, ontology reasoning is not affected and languages with a low reasoning cost such as DL-Lite can be used.

Handling exceptions Since mapping programs are an extension of answer set programming, they inherit the ability to express default rules [31, 22]. This allows a robust handling of data with exceptions or incomplete data: Default values can be attributed to data entries lacking information. When using classical OBDA mappings, the introduction of a new exception requires a rewriting of the mappings to take this new exception into consideration. This is not necessary with mapping programs, as exceptions no longer need to be listed explicitly in each mapping, simplifying maintenance of OBDA mappings.

Example 4. Suppose one has a table `SC_TABLE((<NAME>, <CLEARANCE-CODE>)` representing members in an organization where the first column refers to an individual's name and the second columne refers to a code for their security clearance. The mapping rule

$$\texttt{SC\_TABLE(X, Y)}, \textbf{not}\ \mathsf{Cleared}(\texttt{X}) \rightarrow \mathsf{NotCleared}(\texttt{X})$$

together with the ontology $\{\mathsf{Cleared} \sqsubseteq \neg\mathsf{NotCleared}\}$ assigns by default each individual to the concept $\mathsf{NotCleared}$ unless their clearance is otherwise specified; in other words, it specifies the default rule "individuals usually do not have security clearance". Exceptions to this rule can be added without the need to change this rule, e.g., by adding the mapping rule

$$\texttt{SC\_TABLE(X, "TS")} \rightarrow \mathsf{Cleared}(\texttt{X}).$$

Ontology constraints over the data When constructing an OBDA system, an information manager encodes their knowledge about the how data and ontologies should be translated into the mappings. When using classical OBDA mappings, much of this knowledge must be left implicit; one cannot formulate constraints over the data using the ontology. Being able to formulate these constraints in a logically sound manner has the potential to greatly improve querying performance and mapping maintenance. As a case in point, as mentioned in the discussion after Example 2 of Section 1.1, query rewriting will in general result in an exponential blow-up in size. In the presence of extra knowledge, such as that every Person must be included in a specific table, such growth in query size is redundant. The pruning approaches adopted by existing OBDA implementations [18, 32] prevent this blow-up in certain circumstances, but do not support checking whether this constraint is satisfied by the system; they essentially only prevent the ontology rewriting of specific queries. Thus, in this scenario complete query answering is no longer guaranteed.

Mapping programs introduced in [P1] support such constraints. This allows, for instance, an information manager to state that every Person must be included in a specific table. More importantly, if a Person is found who is not from the specified table, the OBDA specification will be inconsistent. This approach allows for automatic sanity checks, i.e., the ontology and data work together as intended according to the mappings and constraints.

2.2 Ontology Templates

The idea of using macros or templates in order to simplify repetitive tasks is by no means new. This concept has been applied in a multitude of domains, be it within programming languages or even within ontology engineering. As discussed in Section 1.1, various ontology specification tools such as Tawny-OWL [25] and OPPL [7] support the naming of recurring patterns as well as instantiating these with suitable names.

Description Logic Foundation

Reasonable ontology templates were designed to support definition and instantiation of patterns, while maintaining a formal syntax and semantics familiar to users. Thus, the templating framework is first described within the context of description logics.

[P2] introduces *ontology templates* (or templates for short) as named, parameterized ontologies. In other words, an ontology template is an ontology $\mathcal{O}$ with a given *name* and list of designated concept, role, or individual names called *parameters*. Templates can be *instantiated* by substituting each occurrence of its parameters with type-compatible expressions. It is worth noting here that substitutions in ontology templates not only substitute names for names, but can substitute names for complex expressions, allowing for more sophisticated reuse of patterns, as demonstrated in the following example.

Example 5. A template describing a simple part-of relationship can be defined as follows:

$$\mathsf{PartOf}(\mathsf{Part}, \mathsf{Whole}) :: \{\mathsf{Whole} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Part}\}$$

where PartOf is the template's name and Part and Whole are its parameters. Then the instance $\mathsf{PartOf}(\mathsf{Handle}, \mathsf{Hammer})$ yields the ontology

$$\{\mathsf{Hammer} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Handle}\}$$

The instantiation of a template is not limited to basic concepts. Thus PartOf can be instantiated with complex concepts as its arguments. For example, the instance $\mathsf{PartOf}(\mathsf{Grip}, \exists \mathsf{hasPart}.\mathsf{Handle})$, yields the ontology

$$\{\exists \mathsf{hasPart}.\mathsf{Handle} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Grip}\}.$$

Furthermore, templates can be nested, i.e., one template's pattern can contain an instance of another template. An instance of template T then yields an ontology via an expansion procedure where each template instance within the pattern of T is expanded individually. To ensure this terminates one requires that the dependency graph between templates contains no cycles.

Example 6. Consider the template

$$\mathsf{ToolWithPart}(\mathsf{Name}, \mathsf{Part}) :: \{\mathsf{Name} \sqsubseteq \mathsf{Tool}, \mathsf{PartOf}(\mathsf{Part}, \mathsf{Name})\}.$$

Then the instance $\mathsf{ToolWithPart}(\mathsf{Hammer}, \mathsf{Handle})$ would expand to the ontology

$$\{\mathsf{Hammer} \sqsubseteq \mathsf{Tool}, \mathsf{Hammer} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Handle}\}.$$

Though it is quite simple, this formalism provides a solid basis for a variety of very powerful ontology specification and maintenance tools. Since ontology templates are simply named and parameterized DL ontologies they inherit description logic semantics. As such, templates can be used as queries as well as macros. Thus the query results from one template can be used to instantiate another. In other words, we are able to describe certain types of dependencies between patterns within an ontology. This is further explored in [P6], the contributions of which are discussed in Section 2.3.

$$\begin{aligned}
\mathsf{Margherita} &\sqsubseteq \mathsf{NamedPizza}\\
\mathsf{Margherita} &\sqsubseteq \exists \mathsf{hasCountryOfOrigin.Italy}\\
\mathsf{Margherita} &\sqsubseteq \forall \mathsf{hasTopping.Tomato} \sqcup \mathsf{Mozzarella}\\
\mathsf{Margherita} &\sqsubseteq \exists \mathsf{hasTopping.Tomato}\\
\mathsf{Margherita} &\sqsubseteq \exists \mathsf{hasTopping.Mozzarella}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Grandiosa} &\sqsubseteq \mathsf{NamedPizza}\\
\mathsf{Grandiosa} &\sqsubseteq \forall \mathsf{hasTopping.Jarlsberg} \sqcup \mathsf{Ham} \sqcup \mathsf{SweetPepper}\\
\mathsf{Grandiosa} &\sqsubseteq \exists \mathsf{hasTopping.Jarlsberg}\\
\mathsf{Grandiosa} &\sqsubseteq \exists \mathsf{hasTopping.Ham}\\
\mathsf{Grandiosa} &\sqsubseteq \exists \mathsf{hasTopping.SweetPepper}
\end{aligned}$$

(a) Excerpt from DL Pizza ontology

```
<http://draft.ottr.xyz/pizza/NamedPizza> a ottr:Template ;
 ottr:hasParameter
  [ ottr:index 1 ; ottr:classVariable :pizza ] ,
  [ ottr:index 2 ; ottr:individualVariable :country;
      ottr:optional true ] ,
  [ ottr:index 3 ; ottr:listVariable (:toppings) ]  .
   ### body:
[] ottr:templateRef t-owl-axiom:SubClassOf ;
    ottr:withValues ( :pizza p:NamedPizza ) .
[] ottr:templateRef t-owl-axiom:SubObjectHasValue ;
    ottr:withValues ( :pizza p:hasCountryOfOrigin :country ) .
[] ottr:templateRef t-owl-axiom:SubObjectAllValuesFrom ;
    ottr:withValues ( :pizza p:hasTopping _:alltoppings ) .
[] ottr:templateRef t-owl-rstr:ObjectUnionOf ;
    ottr:withValues ( _:alltoppings (:toppings) ) .
[] ottr:templateRef t-owl-axiom:SubObjectSomeValuesFrom ;
    ottr:hasArgument [ ottr:index 1; ottr:value :pizza ] ,
      [ ottr:index 2; ottr:value p:hasTopping ] ,
      [ ottr:index 3; ottr:eachValue (:toppings) ] .
```

(b) wOTTR serialisation of the NamedPizza template.

```
NamedPizza(?Name : 1 class, ?Country : ? individual, ?Toppings : + class)
  :: SubClassOf(?Name, :NamedPizza),
     SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country),
     SubObjectAllValuesFrom(?Name, :hasTopping, _:b1),
     ObjectUnionOf(_:b1, ?Toppings),
     x | SubObjectSomeValuesFrom(?Name, :hasTopping, ?Toppings) .

NamedPizza(:Margherita, :Italy, ⟨:Tomato, :Mozzerella⟩)
NamedPizza(:Grandiosa, none, ⟨:Tomato, :Jarlsberg, :Ham, :SweetPepper⟩)
```

(c) stOTTR serialisation of the NamedPizza template and coresponding instances

```
#OTTR  prefix
p  http://www.co-ode.org/ontologies/pizza/pizza.owl#
#OTTR  end
#OTTR  template  http://draft.ottr.xyz/pizza/NamedPizza
pizza          country   toppings
1              2         3
iri            iri       iri+
p:Margherita   p:Italy   p:Tomato|p:Cheese
p:Grandiosa              p:Tomato|p:Jarlsberg|p:Ham|p:Pepper
#OTTR  end
```

(d) tabOTTR instance serialisation

Figure 2.1: An example of OTTR serializations, adapted from [P4].

Reasonable Ontology Templates (OTTR)

[P2] gives a formal foundation for Reasonable Ontology Templates (OTTR). The core philosophy behind OTTR is to enable the definition and reuse of patterns as well as the maintenance of ontologies as simple and robust as possible [P3, P4]. It is a homo-iconic templating framework for RDF based on ontology templates. In other words, templates for RDF graphs are themselves RDF graphs. In that manner users do not need to learn a new syntax in order to define and reuse patterns; if they so prefer, they can use existing tools and workflows to create OTTR templates.

OTTR strives to separate the modeling and the populating of an ontology. This provides a separation of concerns: ontology engineers need not understand every minute detail of the domain and can rather focus on important relationships and patterns; similarly, domain experts are not required to understand technical details of how the domain is being modeled, and can rather focus on populating an ontology with the correct concepts and properties. As such, we define various serialization formats tailored to various users' expertise: wOTTR (web OTTR) for creating and instantiating templates within RDF; stOTTR (syntax for terse OTTR) for a more human-readable, functional syntax for template creation; and tabOTTR (tabular OTTR) for quick and easy instantiation of templates using a spreadsheet format [P4].

OTTR extends the ontology templates by adding various features: *types* and *cardinalities* to parameters and *expansion modes* to template instances. Typing parameters permits automatically checking whether a template is being used correctly. Giving parameters a cardinality allows for optional or mandatory parameters as well as lists, whereas expansion modes provide multiple options of how to handle list parameters (see [P4] for details).

This extra functionality in addition to its solid formal foundation in ontology templates provides a rich tool-set for maintaining template libraries as well as ontologies built using OTTR. Since templates are logical entities with a clear syntax, one can formally analyze different relationships between templates. This allows, for example, the detection of various forms of redundancy within template libraries and ontologies [P4]. This promises to be a powerful tool for creating and maintaining high-quality ontologies.

Indeed, [P5] provides a starting point for further research in this direction. An empirical use-case study wherein OTTR was applied on a large-scale ontology shows the promise of using formal relations for (semi-)automated redundancy removal. Thus, in [P5] we define the necessary and desirable properties any ontology templating framework must have in order to support interesting formal relations between templates. This is intended as a starting point for further research.

2.3 Rules over Ontologies

OTTR templates are a useful tool for robustly capturing modeling decisions in the ontology specification process: rather than leaving it implicit which patterns are being used, they are explicitly instantiated. Thus, whenever a pattern needs to be changed any ontology using that pattern is automatically updated. This follows the well-known Don't-Repeat-Yourself (DRY) principle. In this section, we discuss how this can be taken even further by introducing rules over the ontology.

As mentioned previously, when combining rules and ontologies the primary focus in research has been on extending the expressivity of a formalism. Our goal in [P6] was not to increase the expressivity of an ontology but rather to create a new framework for explicitly encoding design choices when creating an ontology with templates. More specifically, the motivation is to be able to model the regularity between patterns occurring in an ontology.

This approach provides a high-level view of an ontology: instantiations of patterns are condensed into single instances referring to a specific pattern and relationships between patterns are explicitly described rather than left implicit. On the one hand this promises to compress highly regular ontologies substantially, greatly benefiting readability. On the other hand it provides a more robust framework for maintenance tasks: design choices are explicitly captured and enforced whenever the ontology is altered.

Intuitively, the formalism is based on the dual nature of templates: they can be viewed both as queries and as macros. Thus, the query answers of a template T_B can be used to instantiate another template T_H. Such a pair

$$\mathsf{T}_\mathsf{B} \rightarrow \mathsf{T}_\mathsf{H}$$

is called a *generator*; a set of generators is called a *GBox*. An ontology $\mathcal{O}$ satisfies a generator if for every instance $\mathsf{T}_\mathsf{B}\sigma$ with $\mathcal{O} \models \mathsf{T}_\mathsf{B}\sigma$ then $\mathcal{O} \models \mathsf{T}_\mathsf{H}\sigma$. Here σ is a substitution of the parameters in T_B. The semantics of GBoxes can then be defined analogously to Datalog via a fixpoint operator. However, in order to ensure the existence of a finite model, one must restrict what expressions can be substituted for parameters.

Example 7. Consider the ontology $\mathcal{O} = \{\mathsf{A} \sqsubseteq \exists \mathsf{R}.\mathsf{B}\}$ and the generator $g : \mathsf{T_B} \to \mathsf{T_H}$ for $\mathsf{T_B}(X,Y) :: \{X \sqsubseteq \exists \mathsf{R}.Y\}$ and $\mathsf{T_H}(X,Y) :: \{X \sqsubseteq \exists \mathsf{R}.\exists \mathsf{R}.Y\}$. Then (A,B) is a query answer to $\mathsf{T_B}$ over $\mathcal{O}$, hence g would add $\{\mathsf{A} \sqsubseteq \exists \mathsf{R}.\exists \mathsf{R}.\mathsf{B}\}$ to $\mathcal{O}$. This generates another query answer for $\mathsf{T_B}$, namely $(\mathsf{A}, \exists \mathsf{R}.\mathsf{B})$, thus generating an infinite chain. By restricting the language used to substitute parameters one can avoid this situation and ensure a finite model [P6].

Generators are well suited for describing regularity between ontology patterns. Consider, as an example, an ontology describing a taxonomy of animals where each subclass of Animal is a type of animal. The sentiment "each child of an animal is of the same type as its parents" is not expressible in description logics. Indeed, it must be explicitly written out for any Animal one wishes to include in the ontology. In particular, if another Animal is added (or discovered by reasoning), the rule will not apply. Generators alleviate this issue: By defining two templates

$$\mathsf{T_1(X)} :: \{\mathsf{X} \sqsubseteq \mathsf{Animal}\}$$
$$\mathsf{T_2(Y)} :: \{\mathsf{Y} \sqsubseteq \forall \mathsf{hasChild.Y}\}$$

this design choice can be encoded via the generator $\mathsf{T_1(X)} \to \mathsf{T_2(X)}$. In that manner, the rule will apply to any subclass of Animal and the appropriate template instances will be added whenever another Animal is added.

Furthermore, we define a semantics for GBoxes which incorporates negation-as-failure in generator's bodies. Analogously to other rule-based logics such as Datalog, this results in non-monotonic behavior with multiple possible models. In order to ensure a unique expansion of a GBox over an ontology, we define suitable notions of semipositive and stratifiable GBoxes where a unique expansion can be distinguished.

2.4 Overview of Research Papers

This section gives a brief overview of the papers contained in this dissertation. The relationships between the papers are illustrated in Figure 2.2

Paper 1: Mapping Data to Ontologies With Exceptions Using Answer Set Programming

Authors: Daniel P. Lupp and Evgenij Thorstensen.
In: Proceedings of Norsk Informatikkonfernase (NIK 2018) [P1]. A preliminary version of this paper was presented at the Workshop for Ontologies and Logic Programming for Query Answering 2016.

Abstract In ontology-based data access (OBDA), databases are connected to an ontology via mappings from queries over the database to queries over the ontology. In this paper, we define an ASP-based semantics for mappings from relational databases to first-order ontologies, augmented with queries over the ontology in the mapping rule bodies. The resulting formalism can be described as "ASP modulo theories", and can be used to express constraints and exceptions in OBDA systems, as well as being a powerful mechanism for succinctly representing OBDA mappings. Furthermore, we show that brave reasoning in this setting has either the same data complexity as ASP, or is at least as hard as the complexity of checking entailment for the ontology queries. Moreover, despite the interaction of ASP rules and the ontology, most properties of ASP are preserved. Finally, we show that for ontologies with UCQ-rewritable queries there exists a natural reduction from our framework to ASP with existential variables.

Paper 2: Reasonable Macros for Ontology Construction and Maintenance

Authors: Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen.
In: Proceedings of the 30th International Workshop on Description Logics (DL 2017) [P2].

Abstract Creating and maintaining ontology knowledge bases are difficult processes that can be improved by using macro or templating languages that help structure the ontology engineering task and reduce unnecessary repetitions of ontology patterns. However, since the templates themselves need to be created and maintained, suitable tool support for their maintenance is vital in order to ensure the quality of the resulting knowledge base, and to lower the cost of its construction and maintenance. In this paper, we show that a simple and powerful macro or templating language for description logic (DL) knowledge bases can be defined in description logic itself. In other words, DL allows for macros that are themselves DL knowledge bases; maintenance and debugging for such macros can therefore be done using existing DL reasoners, and does not require extra tool support. We define such macros for the DL $\mathcal{SROIQ}$, which underlies the W3C standard OWL 2. We then show that notions of containment and other problems of interest for such macros become standard reasoning problems supported by existing reasoners. We explore the uses of such macros, showcase how they can be used as restricted higher-order queries, and apply our insights to the setting of data exchange.

Paper 3: Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates

Authors: Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler
In: Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) [P3].

Abstract Reasonable Ontology Templates, OTTRs for short, are OWL ontology macros capable of representing ontology design patterns (ODPs) and closely integrating their use into ontology engineering. An OTTR is itself an OWL ontology or RDF graph, annotated with a special purpose OWL vocabulary. This allows OTTRs to be edited, debugged, published, identified, instantiated, combined, used as queries and bulk transformations, and maintained—all leveraging existing W3C standards, best practices and tools. We show how such templates can drive a technical framework and tools for a practical, efficient and transparent use of ODPs in ontology design and instantiation. The framework allows for a clear separation of the design of an ontology, typically managed by ontology experts, and

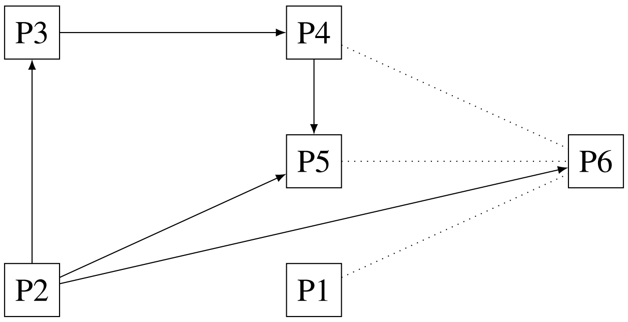

Figure 2.2: This figure depicts the relationships between the papers in this dissertation. A solid arrow represents that a paper directly builds on its predecessor, whereas a dotted line represents related topics or approaches.

its bulk content, provided by domain experts. We illustrate the approach by reconstructing the published Chess Game ODP and producing linked chess data.

Paper 4: Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates

Authors: Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell
In: Proceedings of the 17th International Semantic Web Conference (ISWC 2018) [P4].

Abstract Reasonable Ontology Templates (OTTR) is a language for representing ontology modeling patterns in the form of parameterized ontologies. Ontology templates are simple and powerful abstractions useful for constructing, interacting with, and maintaining ontologies. With ontology templates, modeling patterns can be uniquely identified and encapsulated, broken down into convenient and manageable pieces, instantiated, and used as queries. Formal relations defined over templates support sophisticated maintenance tasks for sets of templates, such as revealing redundancies and suggesting new templates for representing implicit patterns. Ontology templates are designed for practical use; an OWL vocabulary, convenient serialization formats for the semantic web and for terse specification of template definitions and bulk instances are available, including an open source implementation for using templates. Our approach is successfully tested on a real-world large-scale ontology in the engineering domain.

Paper 5: Making a Case for Formal Relations over Ontology Patterns

Authors: Daniel P. Lupp, Leif Harald Karlsen, and Martin G. Skjæveland
In: Proceedings of the 9th Workshop on Ontology Design and Patterns (WOP 2018) [P5].

Abstract There have recently been multiple frameworks proposed to formalize the definition and instantiation of recurring patterns for ontology construction and maintenance. Such formal frameworks can also provide the means necessary for discussing how such patterns can be related to one another, both syntactically and semantically. This has the potential for organizing pattern libraries, robust handling of maintenance tasks, such as redundancy removal, and defining heuristics for what constitutes a “good” pattern. This short paper aims to provide a common ground for discussions on formal relations between ontology patterns. We discuss interesting relations with motivating examples as well as state open questions concerning relations for optimizing the creation, instantiation, and maintenance of ontology patterns

Paper 6: Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity

Authors: Henrik Forssell, Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen
In: Proceedings of the 31st International Workshop on Description Logics (DL 2018) [19]. The version included in Part II is an extended technical report including additional examples and proofs [P6].

Abstract We present a second-order language that can be used to succinctly specify ontologies in a consistent and transparent manner. This language is based on ontology templates (OTTR), a framework for capturing recurring patterns of axioms in ontological modeling. The language, and our results are independent of any specific DL. We define the language and its semantics, including the case of negation-as-failure, investigate reasoning over ontologies specified using our language, and show results about the decidability of useful reasoning tasks about the language itself. We also state and discuss some open problems that we believe to be of interest.

Papers not included in this thesis

Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler. “Reasonable Ontology Templates: APIs for OWL.” in: *International Semantic Web Conference (Posters, Demos & Industry Tracks)*. Vol. 1963. CEUR Workshop Proceedings. CEUR-WS.org, 2017

Martin G. Skjæveland, Leif Harald Karlsen, and Daniel P. Lupp. “Practical Ontology Pattern Instantiation, Discovery, and Maintanence with Reasonable Ontology Templates - Demo paper.” In: *International Semantic Web Conference (P&D/Industry/BlueSky)*. Vol. 2180. CEUR Workshop Proceedings. CEUR-WS.org, 2018

Chapter 3

Conclusion and Future Work

The work presented in this thesis aims to address difficulties that arise during the creation and maintenance of ontologies and OBDI systems. The main focus lies in providing logical frameworks that simplify the specification of ontologies while simultaneously enabling robust, semi-automatic handling of maintenance tasks. To this end, we define a new mapping framework that supports proper handling of incomplete data and exceptions within the mappings, allowing the ontology to be data-agnostic. Furthermore, we propose a new framework for ontology specification, OTTR, which supports the definition, instantiation, and analyzing of ontology patterns. This provides a high-level view of the ontology: rather than solely being able to inspect an ontology on the level of axioms, this additionally enables viewing what is being modeled as opposed to how. This provides a separation of concerns: designing and modeling within an ontology as opposed to populating an ontology with the desired concepts and relationships. Finally, we define an extension of Datalog designed to operate on ontologies. Rather than use rules to enrich an ontology's expressivity, they are designed in order to formally specify certain design choices made during ontology creation. As such, they describe the relationships between patterns within an ontology. In addition to OTTR this serves to provide an even more abstract view of the ontology and what design choices were made.

In the following we describe possible directions of future work.

Mapping programs and query rewritability The complexity of mapping programs is, in general, very high ($\text{NP}^{\mathcal{O}}$-complete data complexity for an ontology with a reasoning oracle $\mathcal{O}$ such that entailment is $|\mathcal{O}|$-complete). This is not feasible for large amounts of data, hence determining fragments with more manageable complexity would be very useful. Furthermore, it is not clear how query rewriting akin to classical OBDA would work in this context. A possible approach could be to restrict mapping programs to positive programs only, and define a suitable notion of stratification on a program's queries. This and other approaches should be investigated in order to ensure practical usability of the framework.

Template relations In [P4] and [P5] we show that defining simple relations such as dependency between templates provides tools for ontology and template maintenance. Interesting questions for further research are (1) to find more sophisticated methods for the detection and removal of various forms of redundancy, such as those discussed in [P5]; (2) what other tasks, such as ontology and template exploration, can benefit from knowledge about the relations between templates; and (3) which more complex relations are of interest, e.g., relations taking types and cardinalities of parameters into account.

OTTR extensions The current implementation of OTTR is entirely syntactic. In particular, running a template as a query over an ontology returns only exact syntactic matches to its pattern. However, a template instance can be entailed by the ontology, but not be syntactically contained. A natural extension to OTTR would be support semantic querying for templates. This would allow for (1) an implementation of GBoxes in the OTTR framework, and (2) more fine-grained analysis of (in particular semantic) relationships between templates.

Existence of a finite GBox expansion As discussed in Section 2.3, the expansion of a GBox is not guaranteed to be finite. In [P6] we avoid this issue by suitably restricting the substitutions of parameters. However, even for simple (e.g., DL-Lite) ontologies it appears to be undecidable to determine whether a finite expansion exists for a given GBox. Thus, it would be useful to (1) develop sophisticated methods to analyze the decidability of a finite expansion given an ontology and a GBox and (2) provide sufficient conditions that guarantee the existence of a finite GBox expansion.

GBox query answering Currently, query answering over an ontology and GBox requires the construction of the expansion. As this can be a very expensive (even undecidable) task, it is worth investigating how and when query answers can be determined without the need for computing the entire expansion.

Nonmonotonic GBoxes In [P6] we define a semantics for negation-as-fallure in generators' bodies. The focus was specifically placed on maintaining the uniqueness of an expansion. A major difference between Datalog and Gboxes is that generators fire if the body is entailed by as opposed to contained in an ontology. As a consequence, there is more potential interaction between generators in a GBox than between standard Datalog rules (cf. the definition of *activation* in [P6]). Therefore, in order to capture multiple possible expansions it would be interesting from a theoretical perspective to investigate whether GBoxes can be imbued with variations of the well-founded and stable model semantics.

Chapter 4
Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, eds. *Foundations of Databases: The Logical Level*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201537710.

[2] M. Alviano, F. Calimeri, W. Faber, G. Ianni, and N. Leone. "Function symbols in ASP: Overview and perspectives." In: *NMR—Essays Celebrating Its 30th Anniversary*. College Publications, 2011, pp. 1–24.

[3] Natalia Antonioli, Francesco Castanò, Spartaco Coletta, Stefano Grossi, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Emanuela Virardi, and Patrizia Castracane. "Developing ontology-based data management for the Italian public debt." In: *22nd Italian Symposium on Advanced Database Systems, SEBD 2014*. Universita Reggio Calabria and Centro di Competenza (ICT-SUD), 2014, pp. 353–360. ISBN: 9781634391450.

[4] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. English. United Kingdom: Cambridge University Press, Apr. 2017. ISBN: 9780521695428.

[5] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*. 2004.

[6] 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." In: *J. Autom. Reasoning* 39.3 (2007), pp. 385–429. DOI: 10.1007/s10817-007-9078-x.

[7] Mikel Egaña, Robert Stevens, and Erick Antezana. "Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language." In: Apr. 2008.

[8] T. Eiter, M. Fink, T. Krennwallner, and C. Redl. "Domain expansion for ASP-programs with external sources." In: *Artif. Intell.* 233 (2016), pp. 84–121.

[9] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. "Combining answer set programming with description logics for the Semantic Web." In: *Art. Intel.* 172.12–13 (2008), pp. 1495–1539. ISSN: 0004-3702. DOI: `http://dx.doi.org/10.1016/j.artint.2008.04.002`.

[P6] Henrik Forssell, Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen. "Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity." In: *CoRR* abs/1809.10436 (2018). arXiv: `1809.10436`. URL: `http://arxiv.org/abs/1809.10436`.

[P2] Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen. "Reasonable Macros for Ontology Construction and Maintenance." In: *Description Logics*. Vol. 1879. CEUR Workshop Proceedings. CEUR-WS.org, 2017.

[10] Fabien Garreau, Laurent Garcia, Claire Lefèvre, and Igor Stéphan. "∃-ASP." In: *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 25-27, 2015.* 2015.

[11] M. Gebser, B. Kaufmann, and T. Schaub. "Conflict-driven Answer Set Solving: From Theory to Practice." In: *Artif. Intell.* 187-188 (Aug. 2012), pp. 52–89. ISSN: 0004-3702. DOI: `10.1016/j.artint.2012.04.001`.

[12] M. Gelfond and V. Lifschitz. "The Stable Model Semantics For Logic Programming." In: *ICLP '88*. MIT Press, 1988, pp. 1070–1080.

[13] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. "Satisfiability solvers." In: *Found. Art. Intel.* 3 (2008), pp. 89–134.

[14] Karl Hammar. "Ontology Design Patterns in WebProtege." In: *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015.* Ed. by Serena Villata, Jeff Z. Pan, and Mauro Dragoni. Vol. 1486. CEUR Workshop Proceedings. CEUR-WS.org, 2015. URL: `http://ceur-ws.org/Vol-1486/paper%5C_50.pdf`.

[15] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, eds. *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*. Vol. 25. Studies on the Semantic Web. IOS Press, 2016. ISBN: 978-1-61499-675-0.

[16] Pascal Hitzler and Cogan Shimizu. "Modular Ontologies as a Bridge Between Human Conceptualization and Data." In: *Graph-Based Representation and Reasoning*. Ed. by Peter Chapman, Dominik Endres, and Nathalie Pernelle. Cham: Springer International Publishing, 2018, pp. 3–6. ISBN: 978-3-319-91379-7.

[17] 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.* W3C Member Submission. May 2004. URL: `http://www.w3.org/Submission/SWRL/`.

[18] D. Hovland, D. Lanti, M. Rezk, and G. Xiao. “Enabling SPARQL Queries over Enterprise Relational Data (Extended Version).” In: *preprint* (2015). arXiv:1605.04263v2 [cs.DB].

[19] Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen. “Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity.” In: *Proceedings of the 31st International Workshop on Description Logics (DL).* (Tempe, Arizona, US, Oct. 27–29, 2018). Ed. by Magdalena Ortiz and Thomas Schneider. CEUR Workshop Proceedings 2211. Aachen, 2018. URL: `http://ceur-ws.org/Vol-2211/#paper-22`.

[20] Roman Kontchakov and Michael Zakharyaschev. “An Introduction to Description Logics and Query Rewriting.” In: *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*. Ed. by Manolis Koubarakis, Giorgos Stamou, Giorgos Stoilos, Ian Horrocks, Phokion Kolaitis, Georg Lausen, and Gerhard Weikum. Cham: Springer International Publishing, 2014, pp. 195–244. DOI: `10.1007/978-3-319-10587-1_5`. URL: `https://doi.org/10.1007/978-3-319-10587-1_5`.

[21] Domenico Lembo, Riccardo Rosati, Valerio Santarelli, Domenico Fabio Savo, and Evgenij Thorstensen. “Approaching OBDA Evolution through Mapping Repair.” In: *Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.* 2016.

[22] V. Lifschitz. “What Is Answer Set Programming?” In: *Proceedings of AAAI 2008*. Ed. by D. Fox and C. P. Gomes. AAAI Press, 2008, pp. 1594–1597. URL: `http://www.aaai.org/Library/AAAI/2008/aaai08-270.php`.

[23] Vladimir Lifschitz. “Minimal Belief and Negation as Failure.” In: *Artif. Intell.* 70 (1994), pp. 53–72.

[24] F. Lin and Y. Zhao. “ASSAT: computing answer sets of a logic program by SAT solvers.” In: *Art. Intel.* 157.1–2 (2004). Nonmonotonic Reasoning, pp. 115–137. ISSN: 0004-3702. DOI: `http://dx.doi.org/10.1016/j.artint.2004.04.004`.

[25] Phillip Lord. “The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL.” In: *OWLED*. Vol. 1080. CEUR Workshop Proceedings. CEUR-WS.org, 2013.

[P5] Daniel P. Lupp, Leif Harald Karlsen, and Martin G. Skjæveland. “Making a Case for Formal Relations over Ontology Patterns.” In: *Proceedings of the 9th Workshop on Ontology Design and Patterns (WOP 2018) co-located with the 17th International Semantic Web Conference (ISWC 2018), Monterey, CA, October 9, 2018.* Vol. 2195. CEUR Workshop Proceedings. CEUR-WS.org, 2018.

[P1] Daniel P. Lupp and Evgenij Thorstensen. “Mapping Data to Ontologies with Exceptions Using Answer Set Programming.” In: *Norsk Informatikkonferanse*. Open Journal Systems. 2018. URL: `http://ojs.bibsys.no/index.php/NIK/article/view/499`.

[26] Carsten Lutz, Inanç Seylan, and Frank Wolter. “Ontology-based Data Access with Closed Predicates is Inherently Intractable (Sometimes).” In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI ’13. Beijing, China: AAAI Press, 2013, pp. 1024–1030. ISBN: 978-1-57735-633-2.

[27] B. Motik and R. Rosati. “Reconciling Description Logics and Rules.” In: *J. ACM* 57.5 (June 2010), 30:1–30:62. ISSN: 0004-5411. DOI: 10.1145/1754399.1754403.

[28] Boris Motik, Ulrike Sattler, and Rudi Studer. “Query Answering for OWL-DL with rules.” In: *J. Web Sem.* 3.1 (2005), pp. 41–60. DOI: 10.1016/j.websem.2005.05.001. URL: https://doi.org/10.1016/j.websem.2005.05.001.

[29] Mark A. Musen. “The protégé project: a look back and a look forward.” In: *AI Matters* 1.4 (2015), pp. 4–12. DOI: 10.1145/2757001.2757003. URL: https://doi.org/10.1145/2757001.2757003.

[30] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. “Journal on Data semantics X.” In: *Journal on Data Semantics X*. Ed. by Stefano Spaccapietra. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 133–173. ISBN: 978-3-540-77687-1.

[31] R. Reiter. “A Logic for Default Reasoning.” In: *Artif. Intell.* 13.1-2 (1980), pp. 81–132. DOI: 10.1016/0004-3702(80)90014-4.

[32] Mariano Rodriguez-Muro and Diego Calvanese. “High Performance Query Answering over DL-Lite Ontologies.” In: *KR*. 2012.

[33] Riccardo Rosati. “Prexto: Query Rewriting under Extensional Constraints in DL - Lite.” In: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*. 2012, pp. 360–374. DOI: 10.1007/978-3-642-30284-8_31. URL: https://doi.org/10.1007/978-3-642-30284-8%5C_31.

[P3] Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler. “Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates.” In: *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017*. 2017.

[34] Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler. “Reasonable Ontology Templates: APIs for OWL.” In: *International Semantic Web Conference (Posters, Demos & Industry Tracks)*. Vol. 1963. CEUR Workshop Proceedings. CEUR-WS.org, 2017.

[35] Martin G. Skjæveland, Leif Harald Karlsen, and Daniel P. Lupp. “Practical Ontology Pattern Instantiation, Discovery, and Maintanence with Reasonable Ontology Templates - Demo paper.” In: *International Semantic Web Conference (P&D/Industry/BlueSky)*. Vol. 2180. CEUR Workshop Proceedings. CEUR-WS.org, 2018.

[P4] Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. “Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates.” In: *The Semantic Web – ISWC 2018*. Ed. by Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl. Springer International Publishing, 2018, pp. 477–494. ISBN: 978-3-030-00671-6.

Part II
Published Articles

Paper 1

Mapping Data to Ontologies with Exceptions Using Answer Set Programming

Daniel P. Lupp and Evgenij Thorstensen. “Mapping Data to Ontologies with Exceptions Using Answer Set Programming.” In: *Norsk Informatikkonferanse*. Open Journal Systems. 2018. URL: `http://ojs.bibsys.no/index.php/NIK/article/view/499` [P1]

Mapping Data to Ontologies With Exceptions Using Answer Set Programming

Daniel P. Lupp and Evgenij Thorstensen

Abstract

In ontology-based data access (OBDA), databases are connected to an ontology via mappings from queries over the database to queries over the ontology. In this paper, we define an ASP-based semantics for mappings from relational databases to first-order ontologies, augmented with queries over the ontology in the mapping rule bodies. The resulting formalism can be described as "ASP modulo theories", and can be used to express constraints and exceptions in OBDA systems, as well as being a powerful mechanism for succinctly representing OBDA mappings. Furthermore, we show that brave reasoning in this setting has either the same data complexity as ASP, or is at least as hard as the complexity of checking entailment for the ontology queries. Moreover, despite the interaction of ASP rules and the ontology, most properties of ASP are preserved. Finally, we show that for ontologies with UCQ-rewritable queries there exists a natural reduction from our framework to ASP with existential variables.

1 Introduction

Ontology-based data access (OBDA) [25] is a method for data integration, utilizing a semantic layer consisting of an ontology and a set of mappings on top of a database. An ontology is a machine-readable model designed to faithfully represent knowledge of a domain independently of the structure of the database; it is comprised of concepts and relationships between these concepts. These ontologies are often formulated using description logics (DLs), a class of decidable logics, due to their desirable computational properties [6].

With the help of mappings, users' queries over the ontology are rewritten into a query over the database language, such as SQL, which can then be run on the source data. To ensure that this rewriting is always possible, one requires the ontology to be first-order rewritable (FOL-rewritable); that is, that every allowed query is equivalent to a first-order formula. However, not all description logics have this property. A common class of ontology languages used in OBDA is the DL-lite family. These description logics have been tailored towards FOL-rewritability and tractable query answering, making them ideally suited for OBDA [6].

Unfortunately, the rewriting step can cause a worst-case exponential blow-up in query size [6]. While this blow-up is necessary to ensure complete query answering, it can lead to highly redundant database queries. Robust pruning of redundant

This paper was presented at the NIK-2019 conference; see http://www.nik.no/.

queries without nonmonotonic features such as extensional constraints [27] or closed predicates [23] is practically infeasible. Furthermore, mapping design and maintenance is usually manual work [2]. This can be a very laborious task, and recent work on mapping evolution and repair [19] attempts to alleviate some of the difficulties involved. However, currently OBDA mappings are interpreted as first-order implications. As a consequence, they lack the expressivity to succinctly handle these issues: for instance, exceptions must be stated explicitly, possibly in multiple mapping assertions.

Example 1 *Let* `TABLE1(<ID>,<DATA>,<CONF>)` *be a table whose third column indicates a measure of confidentiality of the given entry, ranging from 1 (highly confidential) to 5 (not confidential). Furthermore, let $A_C \sqsubseteq A$ be an axiom in the ontology $\mathcal{T}$, where A_C represents the confidential individuals in A. The mapping assertions*

$$\forall X, Y \exists Z : \texttt{TABLE1(X,Y,Z)} \wedge Y = \text{“}a\text{”} \wedge (Z \neq \text{“1”} \wedge Z \neq \text{“2”} \wedge Z \neq \text{“3”}) \rightsquigarrow A(X)$$
$$\forall X, Y \exists Z : \texttt{TABLE1(X,Y,Z)} \wedge Y = \text{“}a\text{”} \wedge (Z = \text{“1”} \vee Z = \text{“2”} \vee Z = \text{“3”}) \rightsquigarrow A_C(X)$$

express that any entry whose `DATA` *column contains "a" is a member of the concept A or A_C depending on the confidentiality level. Changing what constistutes "confidential" to for instance "any data with confidentiality level 2 or above" can represent a major challenge for mapping maintenance: exceptions are listed explicitly and therefore must be changed in every relevant mapping assertion. Such code lists (exceptions depending on the value of a given column) are a common practice in database applications, yet are very prone to error since (1) changes must be made to potentially many mapping assertions and (2) it is entirely manual work, without robust consistency checking.*

Allowing for negation-as-failure as well as ontology queries in the mapping bodies alleviates these issues:the following mapping rules map the data in the same manner, yet have a more succicnt and robust manner of handling exceptions.

$$\forall X, Y \exists Z : \texttt{TABLE1(X,Y,Z)} \wedge Y = \text{“}a\text{”}, \textbf{not}\, A_C(X) \rightarrow A(X)$$
$$\forall X, Y \exists Z : \texttt{TABLE1(X,Y,Z)} \wedge Y = \text{“}a\text{”} \wedge (Z = \text{“1”} \vee Z = \text{“2”} \vee Z = \text{“3”}) \rightarrow A_C(X).$$

In this paper, we propose a new framework for OBDA mappings called mapping programs where mappings are not interpreted as first-order implications. Instead, they are rules containing a database query as well as (positive and negative) ontology queries in their bodies, allowing for existential quantification in both the body and the head of a rule. The ontology queries in rule bodies are evaluated with respect to both the answer sets of the mapping program and the ontology.

This integration of ontology queries into rules allows our formalism to express ontological epistemic constraints, for example extensional constraints [27] and thus a method of pruning redundant queries. Furthermore, by being able to express default rules, mapping programs serve as a powerful abbreviation tool for mapping maintenance (cf. Example 1). This enables the addition of nonmonotonic features to OBDA while retaining the desirable complexity of ontology reasoning. Furthermore, the semantics for OBDA with mapping programs is capable of capturing both the open-world reasoning of the ontology as well as the closed-world reasoning of the database. This was previously not possible with classical mappings, as they are interpreted as first-order theories and thus are inherently open-world.

Related work

Current research on extending OBDA with nonmonontonic capabilities has focused on the ontology side, e.g., through modal description logics or by inclusion of closed predicates [8, 23]. However, the modal semantics can be quite unintuitive. In this setting, modal ontology axioms do not behave well in the presence of nonmodal axioms. Furthermore, extending ontologies with closed predicates quickly results in intractability, cf. [23].

Using a rule-based framework for mappings in OBDA is no new notion; indeed, [3] considers mappings as Datalog programs (possibly with stratified negation) rather than first-order implications. However, this and to our knowledge all previously proposed mapping frameworks are monotonic and thus suffer from the issues illustrated in Example 1.

Since our goal is to connect data to an ontology, we require that each mapping rule contains a database query acting as a guard on the rule. Thus, existential witnesses generated by rules are not further propagated by the mapping program. This is in contrast to the more general existential rules frameworks of tuple-generating dependencies, where existentials in heads of rules may propagate [5, 4]. The decidability of mapping program reasoning therefore reduces entirely to decidability of ontology reasoning.

There have been several approaches to combining rule-based formalisms and description logic ontologies in contexts other than data transformation, be it by constructing a hybrid framework integrating both rules and ontology axioms into the same semantics [24] or by adding rules "on top" [10] of ontologies in the form of DL-programs. Here, rules can interact with an existing knowledge base by including special ontology queries in the rule bodies.

Both DL-programs and mapping programs are special cases of a more general framework called HEX programs [11]. These contain, in addition to regular atoms, queries to external sources in rule bodies which are evaluated with the help of oracles. While extending HEX programs with existential variables in the heads and negative bodies of rules, mapping programs restrict external source queries to database queries (in the positive body of rules, at least one such query must be present in each rule) and ontology queries (in the positive and negative body of rules). These restrictions guarantee that mapping programs are very well-behaved as opposed to general HEX-programs [9]: Despite existential quantification in rules, the presence of database queries ensures very limited and nonrecursive value invention (introduction of new constants), guaranteeing a finite grounding of every mapping program. This provides a solid foundation for query answering and data transformation, as it guarantees termination.

Paper overview

In the following, we define and analyze the general mapping program framework. We discuss the complexity of reasoning in our framework ($\mathrm{NP}^{\mathcal{O}}$-complete, if there exists an ontology reasoning oracle $\mathcal{O}$). Thus, despite the seemingly self-referential definition of satisfiability in mapping programs,[1] mapping program reasoning separates entirely, i.e., mapping programs can be described as "ASP modulo theories" [17] where for most reasoning tasks the ontology is treated as an external

[1]Entailment of rule bodies depends on both the ontology and the answer sets of the mapping program.

black box. As a result, many results from classical ASP, e.g., properties of stratified programs, are directly applicable to this setting. Finally, we consider a special case where the body ontology queries are UCQ-rewritable with respect to the ontology. Using this property, mapping programs can be reduced to classical ASP in a straightforward manner.

2 Preliminaries

OBDA Mappings

Let $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{S}}$ be disjoint signatures containing *ontology predicate symbols*, and *source predicate symbols* respectively. Furthermore, let $\mathcal{C}$ be a set of constants. A *source schema* $\mathcal{S}$ is a relational schema containing relational predicates in $\Sigma_{\mathcal{S}}$ as well as integrity constraints. A *legal database instance* $\mathcal{D}$ over $\mathcal{S}$ is a set of ground atoms from $\Sigma_{\mathcal{S}}$ and $\mathcal{C}$ that satisfies all integrity constraints in $\mathcal{S}$. A first-order formula with free variables is called a *query*, if it has no free variables it is called a *boolean query*. An *ontology* $\mathcal{T}$ is a set of first-order formulas over $\Sigma_{\mathcal{T}}$. In practice, description logics are often used to express ontologies. Thus, though the results in this paper focus on the general case of FOL ontologies, we will use common DL notation throughout the examples in this paper for notational convenience [6].

Example 2 *The axiom $Boss \sqsubseteq \exists hasSup^{-}$ is equivalent to the first-order formula $\forall x(Boss(x) \rightarrow \exists y.hasSup(y,x))$. Here, $hasSup^{-}$ refers to the* inverse role *of $hasSup$.*

Following [18], an OBDA specification is a tuple $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ consisting of a database instance $\mathcal{D}$ legal over a schema $\mathcal{S}$, a FOL-rewritable ontology $\mathcal{T}$ and a set $\mathcal{M}$ consisting of *mapping assertions* of the form $m : \varphi \rightsquigarrow \psi$, where φ and ψ are queries over the data source and ontology, respectively. Then a model $\mathcal{I}$ of an OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ is a first-order model over $\Sigma_{\mathcal{T}} \cup \Sigma_{\mathcal{S}} \cup \mathcal{C}$ that satisfies both $\mathcal{T}$ and $\mathcal{M}$. Here we say that a first-order model $\mathcal{I}$ satisfies a mapping $\mathcal{M}$ if $\mathcal{I} \models \psi(\mathbf{t})$ for every mapping assertion $m : \varphi \rightsquigarrow \psi$ and every tuple $\mathbf{t} \in \text{eval}(\varphi, \mathcal{D})$.

Example 3 *Consider a database consisting of precisely one two-column table `JOBS_DB(<NAME>,<JOB>)`. Furthermore, consider the ontology $\mathcal{T} = \{Empl \sqsubseteq Person, Boss \sqsubseteq Person\}$. In the rewriting process, the query $Person(x)$ would be rewritten to $Person(x) \sqcup Empl(x) \sqcup Boss(x)$ while in the unfolding step, each of the above disjuncts would be expanded to a database query using the mapping assertions. For example, if there exist two mapping assertions `JOBS_DB`$(x, \text{“}Accountant\text{”}) \rightsquigarrow Empl(x)$ and `JOBS_DB`$(x, \text{“}IT\text{”}) \rightsquigarrow Empl(x)$, then the disjunct $Empl(x)$ would be unfolded as `JOBS_DB`$(x, \text{“}IT\text{”}) \vee$ `JOBS_DB`$(x, \text{“}Accountant\text{”})$.*

Example 3 demonstrates some of the current shortcomings of classical OBDA mappings: due to its inherent, first-order nature, it is impossible to distinguish between inferred knowledge and knowledge that is explicit in the database. In the above example, in the presence of a mapping assertion `JOBS_DB`$(x, y) \rightsquigarrow Person(x)$ the query $Person(x)$ would have sufficed without any ontology rewriting, since all desired information was contained in one table. However, while some OBDA implementations [16] support manual query pruning, i.e., the user is able to decide which concepts should not be rewritten, this can potentially lead to incomplete query answering, and there is currently no way of formally checking whether it does. Thus, to ensure complete query answering we have a (potentially redundant) worst-case exponential blow-up in query size.

Another issue with the current aproach is how exceptions and a lack of information are dealt with. Currently, one must keep track of exceptions manually by explicitly listing all exceptions to a rule. Furthermore, due to the closed-world assumption (CWA) in the database, a lack of knowledge is interpreted as knowledge itself, e.g., if something is not contained in the `JOBS_DB` table, it is not a *Person*.

Answer Set Programming

Answer set programming (ASP) is a declarative programming paradigm based on the stable model semantics first defined in [14] as a means of handling default negation in a straightforward manner. It has become one of the more popular logic programming paradigms, due to, e.g., computational benefits such as guaranteed termination as compared to resolution in Prolog [20].

An *ASP-program* P s a set of rules of the form $H \leftarrow B_1, \dots, B_m, \textbf{not}\, C_1, \dots, \textbf{not}\, C_n$. with ground atoms H, B_i, and C_j. The *head* of a rule r is $Head(r) = H$ and the *body* consists of a two parts, the negative body $body^-(r) = \{C_1, \dots C_n\}$ and the positive body $body^+(r) = \{B_1, \dots, B_m\}$. The Herbrand base HB_P of a program P is the set of all possible ground atoms using predicate symbols, function symbols and constants occuring in P. Then for a subset $I \subseteq HB_P$, the Gelfond-Lifschitz reduct P^I of P is the set of rules in P after applying the following changes: (1) If $C_i \in I$ for some i, remove the rule (this corresponds to rules that cannot be applied); (2) in all remaining rules, remove the negative clauses $\textbf{not}\, C_i$ (this corresponds to removing all negative clauses that will evaluate to true).

This reduct is a program without any occurence of negation-as-failure. An interpretation $I \subseteq HB_P$ is called a *stable model* or an *answer set* of P if it is a $\subseteq$-minimal model of P^I, i.e., it is $\subseteq$-minimal and satisfies all rules in P^I.

Though the above semantics require ground atoms, i.e., are essentially propositional, ASP programs might also contains variables or function symbols. In this general case where function symbols are allowed, reasoning becomes undecidable [1]. In the function-free case, the first-order ASP programs are usually first grounded to reduce it to the propositional case. The grounded programs can then either be solved directly [13] or, e.g., translated to an instance of the Boolean satisfiability problem (SAT) before being passed on to efficient SAT solvers [22, 15].

3 OBDA Mapping Programs

In this section we introduce the syntax and semantics for a new framework for OBDA mappings called *mapping programs*. These programs consist of rules that, intuitively, map database queries $Q^{\mathcal{S}}$ to ontology queries $H^{\mathcal{T}}$ provided that certain conditions J^+ and J^- are met. Thus, mapping programs extend classical OBDA mappings with default reasoning.

A *mapping rule* is a rule of the form

$$H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow J_1^+(\mathbf{y}_1'), \dots, J_l^+(\mathbf{y}_l'), \textbf{not}\, J_1^-(\mathbf{y}_1), \dots, \textbf{not}\, J_k^-(\mathbf{y}_k), Q^{\mathcal{S}}(\mathbf{x}).$$

where $\mathbf{y}_i, \mathbf{y}_j' \subseteq \mathbf{x}$ for all i, j. Here, the *head* $H^{\mathcal{T}}(\mathbf{x}, \mathbf{z})$ is a first-order formula over $\Sigma_{\mathcal{T}}$ where $\mathbf{z}$ denotes possible existential variables. The *body* of a mapping rule consists of J_i^-, J_j^+, respectively called the *negative* and *positive justifications* and the *source* *query* $Q^{\mathcal{S}}$. Here, J_i^- and J_j^+ are first-order formulas over the language of $\mathcal{T}$, and the *source query* $Q^{\mathcal{S}}$ is a first-order formula over $\Sigma_{\mathcal{S}}$. A set $\mathcal{M}$ of mapping rules is called a *mapping program*.

Example 4 *Consider a database consisting of one table* `Jobs_DB(<NAME>,<JOB>)`. *Let* $\Sigma_{\mathcal{T}} = \{Empl, hasSup, depHeadOf\}$ *with a unary relation* $Empl$ *of employees and two binary relations* $hasSup$ *and* $depHeadOf$*, describing a supervising relation and a department head relation, respectively. The default rule "employees, of whom we do not know that they are the head of a department, have a supervisor" can be expressed through the following mapping:*

$$m_1 : \exists Z.hasSup(X,Z) \leftarrow Empl(X), \textbf{not}\,\exists Y.depHeadOf(X,Y), \texttt{Jobs\_DB}(X,P).$$

Then a *generalized OBDA specification* is a triple $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where $\mathcal{D}$ is a legal database instance over a schema $\mathcal{S}$, $\mathcal{M}$ is a mapping program, and $\mathcal{T}$ is an ontology.

Definition 1 (Skolem program, following [12]) *Let* $\mathcal{M}$ *be a mapping program. The* Skolem rule $sk(m)$ *associated to a rule* $m \in \mathcal{M}$ *is obtained by replacing each existential variable* v *in* $Head(m)$ *by a new Skolem function symbol* $sk_v(s)$*, where* s *is an ordered sequence of universal variables in* $Head(m)$ *. Then the* Skolem program *of* $\mathcal{M}$ *is* $sk(\mathcal{M}) = \{sk(m) \mid m \in \mathcal{M}\}$.

A *mapping interpretation* $\mathcal{A}$ is a consistent subset of $HB_{sk(\mathcal{M})}$, the Herbrand base over the Skolem program $sk(\mathcal{M})$. Such an interpretation is said to *satisfy* or *model* a positive Skolemized mapping rule

$$m : H^{\mathcal{T}}(\mathbf{x}, sk_{\mathbf{z}}(\mathbf{x})) \leftarrow J_1^+(\mathbf{y}_1'), \ldots, J_l^+(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

written $\mathcal{A} \models m$, if it satisfies the head or does not satisfy the body. It satisfies the body of a rule m if the following holds: for every tuple $\mathbf{t} \in \text{eval}(Q^{\mathcal{S}}, \mathcal{D})$, every interpretation I with $I \models \mathcal{T} \cup \mathcal{A}$ satisfies $J_j^+[\mathbf{t}]$ for all $j \leq l$. Here, $\text{eval}(Q^{\mathcal{S}}, \mathcal{D})$ denotes the set of tuples $\mathbf{t}$ that are answers to the query $Q^{\mathcal{S}}$ over $\mathcal{D}$.

Remark 1 *In this framework, the database query* $Q^{\mathcal{S}}$ *acts as a guard on the mapping rule* m*. It is in general a first-order query. Since* $Q^{\mathcal{S}}$ *is interpreted solely over* $\mathcal{D}$*, mapping rules are not applicable to existential witnesses generated by mapping rule heads. In particular, the database query* $\top(\mathbf{x})$ *is a shorthand for every tuple* $\mathbf{x}$ *occuring in the database.*

For brevity, we write $\mathcal{M}$ instead of $sk(\mathcal{M})$ by abuse of notation. Indeed, in the following we only consider the Skolemized mapping program.

An interpretation $\mathcal{A}$ is said to *satisfy* or *model* a positive mapping program $\mathcal{M}$, written $\mathcal{A} \models \mathcal{M}$, if it satisfies all mapping rules contained in $\mathcal{M}$.

Example 5 *Consider the mapping from Example 4. By Skolemizing, we get the mapping program:*

$$hasSup(X, sk_z(X)) \leftarrow Empl(X), \textbf{not}\,\exists Y.depHeadOf(X,Y), \texttt{Jobs\_DB}(X,P).$$

Definition 2 (Partial ground program, following [12]) *The* partial grounding $PG(m)$ *of a mapping rule* m *is the set of all partial ground instances of* m *over constants in* $\Sigma_{\mathcal{D}}$ *for those variables that are not existential variables in the negative justifications. The* partial ground program *of a mapping program* $\mathcal{M}$ *is the set* $PG(\mathcal{M}) = \bigcup_{m \in \mathcal{M}} PG(m)$.

Example 6 *Consider the database and mapping from Examples 4 and 5. If the set of constants occuring in the database is $\{a, b\}$, then $PG(sk(m_1))$ consists of the four mapping rules for $u, v \in \{a, b\}$:*

$$hasSup(u, sk_z(u)) \leftarrow Empl(u), \textbf{not}\,\exists Y.depHeadOf(u, Y), \texttt{Jobs\_DB}(u, v).$$

Remark 2 (Finite partial grounding) *Though the above definition is seemingly identical to that given in [12], the partial ground program of a mapping program is guaranteed to be finite due to the presence of the database guard $Q^{\mathcal{S}}$ in every mapping rule. This guard is evaluated solely over the domain of the database. In contrast, $\exists$-programs are partially ground using Skolem symbols as well, causing the partial ground program to be infinite as soon as any rule contains an existential head variable.*

Definition 3 ($\mathcal{T}$-reduct) *Given an ontology $\mathcal{T}$, define the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ of a partial ground mapping program $PG(\mathcal{M})$ with respect to an interpretation $\mathcal{A}$ as the mapping program obtained from $PG(\mathcal{M})$ after applying the following:*

1. *Remove all mapping rules m where there exists some $i \leq k$ such that $\mathcal{T} \cup \mathcal{A} \models J_i^-$.*
2. *Remove negative justifications from the remaining rules.*

Example 7 *Continuing with our running example, let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf$, $Boss \sqsubseteq \exists hasSup^-\}$. Add the mapping rules $m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b)$ and $m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P)$. Then for $\mathcal{A} = \{\texttt{Jobs\_DB}(a, b), Empl(a), Boss(a)\}$, the rules*

$$hasSup(a, sk_z(a)) \leftarrow Empl(a), \textbf{not}\,\exists Y.depHeadOf(a, Y), \texttt{Jobs\_DB}(a, v).$$

for $v \in \{a, b\}$ are removed in the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ construction, since $\mathcal{T} \cup \mathcal{A} \models \exists Y.depHeadOf(a, Y)$. Then the $\mathcal{T}$-reduct w.r.t. $\mathcal{A}$ consists of all groundings of the following rules:

$$\begin{aligned} hasSup(b, sk_z(b)) &\leftarrow Empl(b), \texttt{Jobs\_DB}(b, Y). \\ Boss(X) &\leftarrow \texttt{Jobs\_DB}(X, b). \\ Empl(X) &\leftarrow \texttt{Jobs\_DB}(X, P). \end{aligned}$$

A mapping interpretation $\mathcal{A}$ is called a $\mathcal{T}$*-answer set* of $\mathcal{M}$ if it is a $\subseteq$-minimal model of the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$. Then a tuple $(\mathcal{I}, \mathcal{A})$ consisting of a first-order model $\mathcal{I}$ and a mapping interpretation $\mathcal{A}$ is a *model of a generalized OBDA specification* $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ if $\mathcal{I} \models \mathcal{T} \cup \mathcal{A}$ and $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$. For a given ontology $\mathcal{T}$, a mapping program $\mathcal{M}$ is said to *entail a formula* φ, written $\mathcal{M} \models_{\mathcal{T}} \varphi$, if every $\mathcal{T}$-answer set of $\mathcal{M}$ entails φ. Similarly, a generalized OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ entails a formula φ, written $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models \varphi$, if every model of $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ entails φ.

Example 8 *It is easily verifiable that the set $\mathcal{A}$ given in Example 7 is in fact a $\mathcal{T}$-answer set. It does not, however, entail $\mathcal{T}$, as the ontology axiom $Boss \sqsubseteq \exists hasSup^-$ is not satisfied. Thus, to obtain a model of the generalized OBDA specification, any model $\mathcal{I}$ must satisfy this axiom, in addition to the assertions in $\mathcal{A}$.*

Remark 3 (Extensional constraints) *Mapping programs are capable of expressing extensional constraints over the OBDA specification, i.e., constraints over the ontology language on the database and mappings [27]. For instance, the extensional constraint $C \sqsubseteq_e D$, which can be intuitively read as "if $C(a)$ is contained in the ABox, then $D(a)$ is contained in the ABox as well." Such a constraint is expressible with the mapping $\bot \leftarrow \textbf{not}\, D(X), C(X), \top(X)$, where $\bot$ is* bottom *and $\top$ is the query* top *of appropriate arity. This guarantees that any $\mathcal{T}$-answer set of $\mathcal{M}$ must satisfy this constraint. Thus, queries containing the disjunction $C \sqcup D$ can be pruned, as querying for C in addition to D yields no new information. It is worth noting that, while this is similar to integrity constraints over the database, it is not entirely the same: the database schema might differ greatly from the structure of the ontology, thus allowing the possibility of describing database constraints on an ontology level.*

Complexity Analysis

In the general case, where the heads and bodies of mapping rules are allowed to contain arbitrary first-order formulas, reasoning over mapping programs is obviously undecidable. Indeed, consider an empty $\mathcal{T}$ and the mapping program $\mathcal{M} = \{R(a) \leftarrow \top, H(x) \leftarrow \varphi, R(x)\}$ for some arbitrary first-order formula φ. Then $\mathcal{M} \models H(a)$ if and only if φ is a tautology, which is known to be undecidable for arbitrary first-order φ. This is summarized in the following theorem.

Theorem 1 *The problem of checking $\mathcal{M} \models A$ for a given mapping program $\mathcal{M}$ and a ground atom A is undecidable.*

Corollary 1 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be a generalized OBDA specification and A be a ground atom. Then the problem of checking $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models A$ is undecidable.*

Now consider the case where $\mathcal{T} = \emptyset$ and $\mathcal{L}$ is the set of all ground atoms over the language of $\mathcal{T}$. In this case, the oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ must only check membership in $\mathcal{A}$, hence it is linear in the size of $\mathcal{A}$. In this case, a partially ground Skolem mapping program is simply a classical ASP program. Therefore, brave reasoning over partially ground Skolemized mapping programs is at least as hard as classical ground ASP, i.e., is NP-hard [20].

More generally, let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas over the signature $\Sigma_{\mathcal{T}}$ such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. In the following we consider mapping programs $\mathcal{M}$ where the queries in rules are formulas from $\mathcal{L}$. Then to construct a $\mathcal{T}$-answer set, we can employ a variant of the guess-and-check algorithm for ASP: By definition, a set $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ if and only if it is a $\subseteq$-minimal model of the $\mathcal{T}$-reduct $\mathcal{M}^{\mathcal{A}}$. Both the construction of $\mathcal{M}^{\mathcal{A}}$ and the satisfiability-checking are done following their respective definitions. For $\subseteq$-minimality, it suffices to check co-satisfiability of $\mathcal{A} \setminus \{a\}$ for every $a \in \mathcal{A}$, since $\mathcal{M}^{\mathcal{A}}$ is a positive program and hence monotonic.

The complexity of the guess-and-check method is dominated by the oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$: indeed, the oracles $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ and co-$\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ (the oracle that checks if a formula in $\mathcal{L}$ is *not* entailed by $\mathcal{T}$) are called a number of times polynomial in the size of $\mathcal{M}$.

More specifically, for a given oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ brave reasoning over mapping programs is $\mathrm{NP}^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Theorem 2 *Let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of a first-order ontology $\mathcal{T}$ and a set of formulas $\mathcal{L}$ over the language of $\mathcal{T}$ such that $\mathcal{T}$-entailment is $|\mathcal{O}_{(\mathcal{T},\mathcal{L})}|$-hard for an oracle*

$\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. Then for a partially ground Skolemized mapping program $\mathcal{M}$ where the head and all justifications are formulas from $\mathcal{L}$, $\mathcal{T}$-answer set existence is $\text{NP}^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Note that, by the preceding theorem, a partially ground Skolemized mapping program satisfying the conditions of Theorem 2 can be rewritten into an ASP program with oracle calls in the rule bodies. Therefore, mapping programs can be considered as "ASP modulo theories." The resulting ASP program, however, bears little resemblance to the original program, as it is the encoding of an $\text{NP}^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$ Turing machine.

Properties of Mapping Programs

In the previous section, we have seen that reasoning with mapping programs separates in the sense that a candidate $\mathcal{T}$-answer set $\mathcal{A}$ can be checked rule by rule using an ontology reasoning oracle. Therefore, many properties obtainable by syntactic restrictions on ASP transfer automatically to mapping programs. As an example, we show that the proof of answer set uniqueness of stratified ASP programs can be directly transferred to the mapping program setting.

Definition 4 *A mapping program $\mathcal{M}$ is said to be* stratified *if there exists a number l (called the* length *of $\mathcal{M}$) such that each query Q contained in a rule $\mathcal{M}$ can be associated to a natural number $v(Q) \leq l$ where for any rule r in $\mathcal{M}$ $v(Head(r)) \geq \max_{Q \in Body^+(r)} v(Q)$, and $v(Head(r)) > \max_{Q \in Body^-(r)} v(Q)$ hold.*

Then the proof of the following theorem is entirely analogous to its classical/DL-program counterparts [26, 10]. Indeed, using the iterative model sematics [26], a stratification gives rise to a perfect model $\mathcal{A}_{\mathcal{M}}$ of $\mathcal{M}$. It is then easily shown that this model must be a $\mathcal{T}$-answer set, and conversely that any $\mathcal{T}$-answer set is in turn a perfect model of $\mathcal{M}$.

Theorem 3 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be a generalized OBDA specification with a stratified mapping program $\mathcal{M}$. Then $\mathcal{M}$ has a unique $\mathcal{T}$-answer set iff $\mathcal{M}$ is satisfiable.*

UCQ-Rewritable Justifications

We now analyze a restriction of mapping programs that admits a natural reduction to classical ASP for query answering and reasoning. To this end, let $\mathcal{T}$ be an ontology over a decidable fragment of first-order logic. We say a formula φ over $\Sigma_{\mathcal{T}}$ is *UCQ-rewritable with respect to $\mathcal{T}$* if the $\mathcal{T}$-rewriting of φ is equivalent to a union of conjunctive queries [7].

Then for a mapping program $\mathcal{M}$ where all justifications are UCQ-rewritable with respect to $\mathcal{T}$, let $\overline{\mathcal{M}}$, called the *$\mathcal{T}$-rewritten program*, denote the mapping program obtained from $\mathcal{M}$ by replacing every justification with its rewriting with respect to $\mathcal{T}$. The $\mathcal{T}$-rewritten program $\overline{\mathcal{M}}$ is equivalent to a program containing only atoms as positive justifications and CQs as negative justifications, by well-known logic program equivalence transformations [21]. By abuse of notation, $\overline{\mathcal{M}}$ will in the following denote this equivalent program.

Let us first establish the connection between mapping programs and $\exists$-ASP. Recall that a mapping rule can be applied to every tuple $\mathbf{t} \in \text{eval}(Q^{\mathcal{S}}, \mathcal{D})$ where $\mathcal{T} \cup \mathcal{A} \models J_i^+[\mathbf{t}]$ for all positive justifications J_j^+ and $\mathcal{T} \cup \mathcal{A} \not\models J_j^-[\mathbf{t}]$ for all negative justifications J_j^-. If the TBox $\mathcal{T}$ is empty, this reduces to checking whether the justifications are certain answers w.r.t. $\mathcal{A}$ and hence simply checking containment in $\mathcal{A}$. This is, however, precisely the semantics of ASP with existential variables. Hence, mapping programs can be seen as an extension of $\exists$-ASP [12], both semantically and syntactically. This result is summarized in the following theorem.

Theorem 4 *Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are conjunctive queries. Then a set $\mathcal{A}$ is a $\emptyset$-answer set of $\mathcal{M}$ iff it is a $\exists$-answer set of $\mathcal{M}$.*

The following lemma describes the relationship between $\mathcal{T}$-rewritten programs and reducts w.r.t. $\mathcal{A}$, which is particularly useful when analyzing the connection between $\exists$-ASP and mapping programs, as discussed in Theorem 5.

Lemma 1 *For any $\mathcal{A} \subseteq HB_{sk(\mathcal{M})}$ we have $\overline{\mathcal{M}}^{\mathcal{A}} = \overline{\mathcal{M}^{\mathcal{A}}}$, where $\overline{\mathcal{M}^{\mathcal{A}}}$ denotes the $\mathcal{T}$-rewritten program of $\mathcal{M}^{\mathcal{A}}$.*

Theorem 5 *Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are UCQ-rewritable with respect to an ontology $\mathcal{T}$. A set $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ iff it is an $\emptyset$-answer set of $\overline{\mathcal{M}}$.*

As a direct consequence of the preceding theorem, the following corollary describes how query answering over an OBDA specification using a UCQ-rewritable mapping program can be reduced to query answering over an equivalent OBDA specification with an empty ontology.

Corollary 2 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be an OBDA specification, $\overline{\mathcal{M}}$ the $\mathcal{T}$-rewritten program of $\mathcal{M}$, q a query over $\Sigma_{\mathcal{T}}$, and $\overline{q}$ its rewriting with respect to $\mathcal{T}$. Then $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff (\mathcal{D}, \overline{\mathcal{M}}, \emptyset) \models \overline{q}[t]$.*

Therefore, by Corollary 2 and Theorem 4 we find that every UCQ-rewritable mapping program $\mathcal{M}$ is equivalent (w.r.t. answer sets) to an $\exists$-ASP program. Then, by results in [12], this can be reduced to a classical ASP program. This is summarized in the following theorem.

Theorem 6 *For an OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where the justifications in $\mathcal{M}$ are UCQ-rewritable with respect to $\mathcal{T}$, there exists an ASP program $\mathcal{M}'$ such that for a query q over $\mathcal{T}$ $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff \mathcal{M}' \models \overline{q}[t]$, i.e., query answering over $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ reduces to cautious reasoning over $\mathcal{M}'$.*

4 Conclusion and Future Work

In this paper, we propose a new mapping framework for ontology-based data access (and data transformation in general) that is both very expressive and controllable with respect to computational complexity. Our framework allows for default reasoning as well as the expression of epistemic properties over the database and ontology. We show that mapping programs can be described as "ASP modulo theories," treating the ontology as an external black box. The loose coupling of ASP and ontological reasoning allows us to transfer existing results about ASP to this setting.

While various highly optimized ASP solvers do exist, the data complexity involved is rather undesirable in the context of real-world OBDA and big data.

Therefore, one of the greatest priorities regarding future work is to determine how and when the complexity can be reduced; the mapping program should not be run on the entire data set. Furthermore, there appears to be a strong connection between mapping programs and satisfiability modulo theories (SMT). We believe that mapping programs with ontologies supported by SMT solvers should be translatable into SMT instances in the same manner as ASP programs can be translated into SAT instances. If true, this could allow for very efficient query answering in certain OBDA settings. As such, investigating this connection is a clear goal for future work.

References

[1] M. Alviano, F. Calimeri, W. Faber, G. Ianni, and N. Leone. Function symbols in ASP: Overview and perspectives. In *NMR—Essays Celebrating Its 30th Anniversary*, pages 1–24. College Publications, 2011.

[2] N. Antonioli, F. Castan, S. Coletta, S. Grossi, D. Lembo, M. Lenzerini, A. Poggi, E. Virardi, and P. Castracane. Developing ontology-based data management for the italian public debt. In *22nd Italian Symposium on Advanced Database Systems, SEBD 2014*, pages 353–360. Universita Reggio Calabria and Centro di Competenza (ICT-SUD), 2014.

[3] R. Barilaro, N. Leone, F. Ricca, and G. Terracina. Distributed ontology based data access via logic programming. In *Proceedings of RR 2012*, pages 205–208, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)*, 48:115–174, 2013.

[5] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[7] F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proceedings of EDBT 2013*, pages 561–572, New York, NY, USA, 2013. ACM.

[8] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Logic*, 3(2):177–225, April 2002.

[9] T. Eiter, M. Fink, T. Krennwallner, and C. Redl. Domain expansion for ASP-programs with external sources. *Artif. Intell.*, 233:84–121, 2016.

[10] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Art. Intel.*, 172(1213):1495 – 1539, 2008.

[11] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of IJCAI 2005*, pages 90–96, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[12] F. Garreau, L. Garcia, C. Lefèvre, and I. Stéphan. $\exists$-ASP. In *Proceedings of JOW 2015 co-located with the IJCAI 2015, Buenos Aires, Argentina, July 25-27,*, 2015.

[13] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187-188:52–89, August 2012.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP '88*, pages 1070–1080. MIT Press, 1988.

[15] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Found. Art. Intel.*, 3:89–134, 2008.

[16] D. Hovland, D. Lanti, M. Rezk, and G. Xiao. Enabling SPARQL queries over enterprise relational data (extended version). *preprint*, 2015. arXiv:1605.04263v2 [cs.DB].

[17] J. Lee and Y. Meng. Answer set programming modulo theories and reasoning about continuous changes. In *Proceedings of the IJCAI 2013*, pages 990–996. AAAI Press, 2013.

[18] D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Mapping analysis in ontology-based data access: Algorithms and complexity. In *Proceedings of ISWC 2015*, pages 217–234, 2015.

[19] D. Lembo, R. Rosati, V. Santarelli, D. F. Savo, and E. Thorstensen. Approaching OBDA evolution through mapping repair. In *Proceedings of DL 2016.*, 2016.

[20] V. Lifschitz. What is answer set programming? In D. Fox and C. P. Gomes, editors, *Proceedings of AAAI 2008*, pages 1594–1597. AAAI Press, 2008.

[21] V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. *Ann. Math. Art. Intel.*, 25(3):369–389, 1999.

[22] F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Art. Intel.*, 157(12):115 – 137, 2004. Nonmonotonic Reasoning.

[23] C. Lutz, I. Seylan, and F. Wolter. Ontology-based data access with closed predicates is inherently intractable (sometimes). In *Proceedings of IJCAI 2013*, pages 1024–1030. AAAI Press, 2013.

[24] B. Motik and R. Rosati. Reconciling description logics and rules. *J. ACM*, 57(5):30:1–30:62, June 2010.

[25] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. In S. Spaccapietra, editor, *J. Data Sem. X*, pages 133–173. Springer-Verlag, Berlin, Heidelberg, 2008.

[26] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[27] R. Rosati. Prexto: Query rewriting under extensional constraints in DLLite. In E. Simperl, P. Cimiano, A. Polleres, O. Corcho, and V. Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 360–374. Springer Berlin Heidelberg, 2012.

Appendix

In this appendix we include the proofs omitted from [P1].

Theorem 2. Let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of a first-order ontology $\mathcal{T}$ and a set of formulas $\mathcal{L}$ over the language of $\mathcal{T}$ such that $\mathcal{T}$-entailment is $|\mathcal{O}_{(\mathcal{T},\mathcal{L})}|$-hard for an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. Then for a partially ground Skolemized mapping program $\mathcal{M}$ where the head and all justifications are formulas from $\mathcal{L}$, $\mathcal{T}$-answer set existence is NP$^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Proof. Membership in NP$^{\mathcal{O}_{\mathcal{T},\mathcal{L}}}$ can be seen via a guess-and-check method. Verifying that a certificate $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ requires (1) constructing the reduct $\mathcal{M}^{\mathcal{A}}$ and (2) checking that $\mathcal{A}$ is the $\subseteq$-minimal model of $\mathcal{M}^{\mathcal{A}}$. Constructing the reduct requires a number of calls to $\mathcal{O}_{\mathcal{T},\mathcal{L}}$ bounded by the number of negative justifications in $\mathcal{M}$. Since $\mathcal{M}^{\mathcal{A}}$ is a positive program its minimal model can be constructed using an iterated fixpoint construction analogous to classical Datalog [1] and compared to $\mathcal{A}$, thus requiring a number of calls to $\mathcal{O}_{\mathcal{T},\mathcal{L}}$ polynomial in the size of $\mathcal{M}^{\mathcal{A}}$.

For hardness, consider an NP$^{\mathcal{O}_{\mathcal{T},\mathcal{L}}}$ oracle machine S with transition function δ. We can encode δ as a mapping program in the same way an NP Turing machine can be encoded as a propositional ASP program (which must exist since propositional ASP is NP-complete): the Turing machine halts in an accepting configuration iff the corresponding program has an answer set. For all transitions that do not result in an oracle call are translated analogously to classical ASP. Let C_1, C_2, C_3 be encodings of configurations of S such that $\delta(C_1) = C_2$ if $\mathcal{O}_{\mathcal{T},\mathcal{L}}(C_1) = \mathsf{TRUE}$ and $\delta(C_1) = C_3$ if $\mathcal{O}_{\mathcal{T},\mathcal{L}}(C_1) = \mathsf{FALSE}$. This transition is captured by the mapping rules

$$\top, C_1 \rightarrow C_2$$
$$\top, \textbf{not}\, C_1 \rightarrow C_3.$$

□

Theorem 3. Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be a generalized OBDA specification with a stratified mapping program $\mathcal{M}$. Then $\mathcal{M}$ has a unique $\mathcal{T}$-answer set iff $\mathcal{M}$ is satisfiable.

Proof. The stratification of $\mathcal{M}$ gives rise to a partitioning $\mathcal{M}_1, \ldots, \mathcal{M}_k$ of $\mathcal{M}$, where each $\mathcal{M}_i$ contains all mapping rules m with $v(Head(m)) = i$. Then the proof of the existence of a perfect model is entirely analogous to the stratified Datalog case: $\mathcal{M}_1$ is a positive program and hence has as a unique minimal model. $\mathcal{M}_2$ is semi-positive with respect to $\mathcal{M}_1$, i.e., the negative justifications within rules in $\mathcal{M}_2$ cannot occur in a head in $\mathcal{M}_2$. Hence $\mathcal{M}_2$ has a unique minimal model. Iterating this argument gives the perfect model $\mathcal{A}_\mathcal{M}$ of $\mathcal{M}$.

By a similar argument, it is easily shown that any $\mathcal{T}$-answer set of $\mathcal{M}$ must contain $\mathcal{A}_\mathcal{M}$. However, $\mathcal{T}$-answer sets cannot properly contain a model of $\mathcal{M}$ (this would contradict the minimality condition of $\mathcal{T}$-answer sets), proving uniqueness. □

Theorem 4. Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are conjunctive queries. Then a set $\mathcal{A}$ is a $\emptyset$-answer set of $\mathcal{M}$ iff it is a $\exists$-answer set of $\mathcal{M}$.

Proof. If $\mathcal{T}$ is empty, checking satisfaction of a justification J^- reduces to checking whether it is a certain answer w.r.t. $\mathcal{A}$, i.e., checking containment in $\mathcal{A}$. This corresponds precisely to the semantics of $\exists$-ASP. □

Lemma 1. For any $A \subseteq HB_{sk(\mathcal{M})}$ we have $\overline{\mathcal{M}}^A = \overline{\mathcal{M}^A}$, where $\overline{\mathcal{M}^A}$ denotes the $\mathcal{T}$-rewritten program of $\mathcal{M}^A$.

Proof. Let m be a mapping rule removed from $\mathcal{M}$ in the construction of the $\mathcal{T}$-reduct $\mathcal{M}^A$, i.e., there exists some $i \leq k$ such that $\mathcal{T} \cup A \vDash J_i^-$. This is equivalent to $\emptyset, A \vDash \overline{J_i^-}$ where $\overline{J_i^-}$ is the $\mathcal{T}$-rewriting of J_i^-. Thus $\overline{m}$ is removed from $\overline{\mathcal{M}}$ in the construction of the $\mathcal{T}$-reduct $\overline{\mathcal{M}}^A$. Hence $\overline{m} \in \overline{\mathcal{M}}^A$ iff $m \in \mathcal{M}^A$ iff $\overline{m} \in \overline{\mathcal{M}^A}$. □

Theorem 5. Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are UCQ-rewritable with respect to an ontology $\mathcal{T}$. A set $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ iff it is an $\emptyset$-answer set of $\overline{\mathcal{M}}$.

Proof. Let $\mathcal{A} \subseteq HB_{sk(\mathcal{M})}$ and let

$$r : H^\mathcal{T}[\mathbf{t}, sk_\mathbf{z}(\mathbf{t})] \leftarrow J_1^+[\mathbf{t}], \ldots, J_k^+[\mathbf{t}]$$

be any rule in $\mathcal{M}^\mathcal{A}$. We shall prove the statement by separately showing (1) the equivalence of rule satisfaction in $\mathcal{M}^\mathcal{A}$ and $\overline{\mathcal{M}}^\mathcal{A}$ and (2) that $\mathcal{A} \vDash \mathcal{M}^\mathcal{A}$ is minimal iff $\mathcal{A} \vDash \overline{\mathcal{M}}^\mathcal{A}$ is minimal.

1. *Satisfaction:*

$$\begin{aligned}
&\mathcal{A} \models r \\
\Leftrightarrow\ &\text{if } \mathcal{A} \models Q^{\mathcal{S}}[\mathbf{t}] \text{ and } \mathcal{A}, \mathcal{T} \models J_i^+[\mathbf{t}] \text{ for all } i \leq k \\
&\text{then } \mathcal{A} \models H^{\mathcal{T}}[\mathbf{t}, sk_{\mathbf{z}}(\mathbf{t})]. \\
\Leftrightarrow\ &\text{if } \mathcal{A} \models Q^{\mathcal{S}}[\mathbf{t}] \text{ and } \mathcal{A}, \emptyset \models \overline{J_i^+}[\mathbf{t}] \text{ for all } i \leq k \\
&\text{then } \mathcal{A} \models H^{\mathcal{T}}[\mathbf{t}, sk_{\mathbf{z}}(\mathbf{t})]. \\
\Leftrightarrow &\mathcal{A} \models \overline{r}
\end{aligned}$$

2. *Minimality:* Assume $\mathcal{A}' \subsetneq \mathcal{A}$ is a model of $\mathcal{M}^{\mathcal{A}}$. By 1, this is the case if and only if $\mathcal{A}' \models \overline{\mathcal{M}^{\mathcal{A}}}$. Finally, Lemma 1 yields the desired result that $\mathcal{A}' \models \overline{\mathcal{M}}^{\mathcal{A}}$.

□

Corollary 2. Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be an OBDA specification, $\overline{\mathcal{M}}$ the $\mathcal{T}$-rewritten program of $\mathcal{M}$, q a query over $\Sigma_{\mathcal{T}}$, and $\overline{q}$ its rewriting with respect to $\mathcal{T}$. Then $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff (\mathcal{D}, \overline{\mathcal{M}}, \emptyset) \models \overline{q}[t]$.

Proof. Direct consequence of Theorem 5. □

Theorem 6. For an OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where the justifications in $\mathcal{M}$ are UCQ-rewritable with respect to $\mathcal{T}$, there exists an ASP program $\mathcal{M}'$ such that for a query q over $\mathcal{T}$ $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff \mathcal{M}' \models \overline{q}[t]$, i.e., query answering over $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ reduces to cautious reasoning over $\mathcal{M}'$.

Proof. By Theorem 4 and Corollary 2, $\mathcal{M}$ is equivalent to an $\exists$-ASP program P. By Proposition 8 in Garreau et al [1] this can further be reduced to a classical ASP program. □

[1]Fabien Garreau, Laurent Garcia, Claire Lefèvre, and Igor Stéphan. "∃-ASP." in: *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 25-27, 2015.* 2015

Paper 2

Reasonable Macros for Ontology Construction and Maintenance

Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen. “Reasonable Macros for Ontology Construction and Maintenance.” In: *Description Logics*. Vol. 1879. CEUR Workshop Proceedings. CEUR-WS.org, 2017 [P2]

Reasonable Macros for Ontology Construction and Maintenance

Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen

Department of Informatics, University of Oslo
`{jonf,danielup,martige,evgenit}@ifi.uio.no`

Abstract. Creating and maintaining ontology knowledge bases are difficult processes that can be improved by using macro or templating languages that help structure the ontology engineering task and reduce unnecessary repetitions of ontology patterns. However, since the templates themselves need to be created and maintained, suitable tool support for their maintenance is vital in order to ensure the quality of the resulting knowledge base, and to lower the cost of its construction and maintenance. In this paper, we show that a simple and powerful macro or templating language for description logic (DL) knowledge bases can be defined in description logic itself. In other words, DL allows for macros that are themselves DL knowledge bases; maintenance and debugging for such macros can therefore be done using existing DL reasoners, and does not require extra tool support. We define such macros for the DL $\mathcal{SROIQ}$, which underlies the W3C standard OWL 2. We then show that notions of containment and other problems of interest for such macros become standard reasoning problems supported by existing reasoners. We explore the uses of such macros, showcase how they can be used as restricted higher-order queries, and apply our insights to the setting of data exchange.

1 Introduction

Like any knowledge representation task, creating ontologies is a difficult and time-consuming process that requires the utmost diligence of the ontology engineer. Ontology design patterns (ODPs) are a proposed solution to improve the ontology engineering process by, in part, providing reusable favourable ontology building blocks to recurrent ontology modelling problems, thus avoiding common pitfalls and introducing best modelling practices [9, 14]. Many tools and languages exist for building ontologies using patterns, macros or templates, e.g., [17, 18, 22, 24, 30]. However, they in turn incur the cost of needing tool support to manage and debug the macros or templates as well as the ontologies themselves. The latter is easier, since OWL is a standard with a lot of tool support available. In fact, to our knowledge, existing ontology construction tools rely completely on the resulting OWL ontology for checking the ontological quality of the templating mechanism, if this is addressed at all.

The goal of our paper is to show that a powerful macro language for ontologies expressed in common description logics (DLs) can be defined using the description logics themselves. In other words, we present the simple, but novel idea of using ontologies to represent macros for description logics, and call such macros *ontology templates* or just *templates*. A template is itself an ontology in the given DL language. Templates may have parameters, may be defined from other templates, and are instantiated by substituting parameters with suitable concept expressions, role expressions, and constants in the ontology. A template defined from other templates may be expanded by recursively replacing template expressions with the ontology they represent. Thus, an expanded template represents the semantics of the pattern in the form of a prototypical ontology.

A set of templates can actively be used in the ontology engineering process as an abstraction interface for representing ontological statements at a suitable level of granularity, i.e., using its unexpanded form. The templates themselves may be constructed and maintained, both in isolation and as an expanded ontology, using available ontology editors and reasoners. One can imagine the design of an ontology relying completely on a (relatively small) set of well-organised templates managed by ontology experts, while the bulk content of the ontology is represented as a (large) set of template instances, typically managed by domain experts.

The construction, maintenance and debugging of templates require little additional tool support, as templates may be built using existing ontology editors, and problems related to redundancy and correctness become standard reasoning problems that are supported by existing ontology reasoners. What does require additional tool support, is the process of instantiating templates. A prototype implementation is available which specifies an OWL vocabulary for expressing templates and instances, and software to interpret the vocabulary and produce expansions, queries, transformations and data input formats from the templates [1].

Example 1. To build some intuitions, we give an informal example before formally defining templates and operations on templates. The following expression $\mathsf{P}(\mathsf{x}, \mathsf{y}) \mapsto \{\mathsf{x} \sqsubseteq \exists R.\mathsf{y}\}$ represents a template

P with parameters x and y. We may instantiate P with concepts C, D, written $\mathsf{P}(C, D)$, to obtain the instance $\{C \sqsubseteq \exists R.D\}$. Using the template P, we can now define the complex template Q as $\mathsf{Q}(\mathsf{x}, \mathsf{y}, \mathsf{z}) \mapsto \{\mathsf{P}(\mathsf{x}, \mathsf{y}), \mathsf{P}(\mathsf{y}, \mathsf{z}), \mathsf{x} \sqsubseteq \forall S.\mathsf{z}\}$. Expanding $\mathsf{Q}(\mathsf{x}, \mathsf{y}, \mathsf{z})$ gives the ontology $\{\mathsf{x} \sqsubseteq \exists R.\mathsf{y}, \mathsf{y} \sqsubseteq \exists R.\mathsf{z}, \mathsf{x} \sqsubseteq \forall S.\mathsf{z}\}$.

Any macro or templating language can also be viewed as a query language, where a macro producing an object given values becomes a query asking for values that would produce a given object. In our case, a template is a knowledge base that takes individual names as well as concept and role expressions as input. Using it as a query over another knowledge base is then a way of extracting from this knowledge base concept and role expressions and individuals of interest. This is both a tool for exploring large ontologies and for ontology transformation and maintenance. Due to the fact that templates are also ontologies, problems such as query containment become standard DL reasoning problems.

By exploiting the duality between templates as macros and templates as queries, we show that templates find natural use in data exchange and ontology approximation. Pairs of templates can be seen as specifying fairly rich mappings between different ontologies. Since templates are themselves ontologies, existing reasoners can be used to good effect in computing properties of interest. We want to further exploit this fact to explore the different relationships between patterns represented as templates which are useful for their maintenance and use, cf. [11].

1.1 Related work

A predecessor and inspiration to the current form of templates was presented in [19]. To our knowledge, the idea of representing macros for description logics as ontologies is not previously discussed in the literature. However, it is related both to various macro approaches in use, as well as to notions from the field of data exchange.

In a paper from 2005 [29], Vrandečić introduces the concept of macros for OWL ontologies and discusses advantages and disadvantages of macros, and their possible applications and implementations. Here, a macro is defined as a symbol, possibly with parameters, and is expanded according to a set of rules. Our papers share many ideas, but Vrandečić's exposition is less detailed; for instance the format of the rules is left unspecified. Also, it is not clear if macros can be composed from other macros, which is a core feature of our templates.

Mappings in the field of data exchange are also related, where queries over source data or knowledge bases are mapped to queries over a target schema or ontology [2, 4, 21]. In a sense, the target queries are used as macros to produce an (incomplete) database or ABox from data retrieved by the source queries. Our approach differs in that templates, when viewed as queries, are a restricted case of higher-order queries, as concept and role expressions are part of the answers. In data exchange, queries are furthermore supposed to produce, as far as possible, a ground database or ABox, as opposed to a full theory in some logical language.

Viewed as queries, templates are a very restricted kind of higher-order queries; they return concept and role expressions in addition to individual names. Higher-order queries for DL knowledge bases have been investigated for OWL [23], but usually with very expressive semantics based on entailment [7]. We recognise the merits of such an approach, but the drawback is that very careful and strong restrictions are required to control decidability and complexity. In our case, the reliance on substitution allows us to bypass these problems.

Many practical tools and languages for constructing knowledge bases using template or macro mechanisms exist; a few of these are the following: R2RML [6], OPPL [17] M^2 [24], XDP [10,12] Ontorat [30], Populous [18], XLWrap [20], RDF123 [13], Tawny-OWL [22], Thea [28] and InfixOWL [25].

2 Preliminaries

We take as our starting point the expressive description logic language $\mathcal{SROIQ}$ that underlies the W3C standard OWL 2 [16]. Many other well-studied and used DL languages (e.g. DL-Lite$_R$ [5] and $\mathcal{EL}_\perp$ [3], which underlie OWL 2 profiles) can be defined in terms of syntactic restrictions on $\mathcal{SROIQ}$. Our definitions of $\mathcal{SROIQ}$ templates will therefore carry over to these languages in a straightforward manner.

$\mathcal{SROIQ}$ uses a vocabulary $N = N_C \cup N_R \cup N_I$ with countable sets of *concept names* N_C, *role names* N_R, and *individual names* N_I. *Complex concepts* and *complex roles* are defined as the smallest sets containing all concepts and roles that can be inductively constructed using the concept and role constructors in Fig. 1, where C, D are concepts, P, R are atomic or inverse roles, S, Q are any roles including role chains, a, b are individual names, and n a positive integer. A $\mathcal{SROIQ}$ *knowledge base* (KB) is a finite set of axioms of the form shown in Fig. 1. Further restrictions apply, so that not every finite set of axioms of this form is a $\mathcal{SROIQ}$ knowledge base[1]. For instance, $S \cdot Q \sqsubseteq R$ and $Dis(P, R)$ is an illegal combination of axioms. See [16] for further details.

Expression	Syntax	Semantics	Subst. app.
Complement	$\neg C$	$\Delta^I \setminus C^I$	$\neg(C\sigma)$
Intersection	$C \sqcap D$	$C^I \cap D^I$	$C\sigma \sqcap D\sigma$
Union	$C \sqcup D$	$C^I \cup D^I$	$C\sigma \sqcup D\sigma$
Exist. restr.	$\exists R.C$	$\{x \mid \exists y \langle x, y\rangle \in R^I \land y \in C^I\}$	$\exists(R\sigma).(C\sigma)$
Univ. restr.	$\forall R.C$	$\{x \mid \forall y \langle x, y\rangle \in R^I \to y \in C^I\}$	$\forall(R\sigma).(C\sigma)$
Card. restr.	$\geq nR.C$	$\{x \mid \#\{y \mid \langle x, y\rangle \in R^I \land y \in C^I\} \geq n\}$	$\geq n(R\sigma).(C\sigma)$
Self restr.	$\exists R.Self$	$\{x \mid \langle x, x\rangle \in R^I\}$	$\exists(R\sigma).Self$
Nominal	$\{a\}$	$\{a^I\}$	$\{a\sigma\}$
Inv. role	R^-	$\{\langle y, x\rangle \mid \langle x, y\rangle \in R^I\}$	$(R\sigma)^-$
Univ. role, conc.	$U, \top$	$\Delta^I \times \Delta^I, \Delta^I$	$U, \top$ (invar.)
Empty conc.	$\bot$	$\emptyset$	$\bot$ (invar.)
Chain (w)	$S \cdot Q$	$S^I \circ Q^I$	$S\sigma \cdot Q\sigma$
Axiom	**Syntax**	**Semantics**	**Subst. app.**
Conc. incl.	$C \sqsubseteq D$	$C^I \subseteq D^I$	$C\sigma \sqsubseteq D\sigma$
Role incl.	$w \sqsubseteq R$	$w^I \subseteq R^I$	$w\sigma \sqsubseteq R\sigma$
Reflexivity assert.	$Ref(R)$	R^I is reflexive	$Ref(R\sigma)$
Role disjointness	$Dis(P, R)$	$P^I \cap R^I = \emptyset$	$Dis(P\sigma, R\sigma)$
Concept assert.	$C(a)$	$a^I \in C^I$	$C\sigma(a\sigma)$
Role assert.	$R(a, b)$	$\langle a^I, b^I\rangle \in R^I$	$R\sigma(a\sigma, b\sigma)$
Neg. role assert.	$\neg R(a, b)$	$\langle a^I, b^I\rangle \notin R^I$	$\neg R\sigma(a\sigma, b\sigma)$

Fig. 1. Syntax, semantics, and substitution application for $\mathcal{SROIQ}$

A *DL interpretation* is a pair $\langle \Delta^I, \cdot^I\rangle$ where $\Delta^I \neq \emptyset$ and $\cdot^I$ is a function such that $a^I \in \Delta^I$ for all $a \in N^I$, $A^I \subseteq \Delta^I$ for all $A \in N_C$, and $R^I \subseteq \Delta^I \times \Delta^I$ for all $R \in N_R$. Interpretation of concept and role expressions is inductively defined as given in Fig. 1. An axiom ϕ is *satisfied* by I if I satisfies the respective constraint in Fig. 1, and a knowledge base K is satisfied by I if I satisfies every axiom in K.

Substitutions. Let L be a description logic language. An *L-substitution* σ is a function from the sets of concept, role, and individual names

[1] Since we do not distinguish between terminological axioms and assertions in this paper, we also use "knowledge base" and "ontology" interchangeably.

to, respectively, the sets of concept expressions, role expressions, and individual names allowed in L. Given a set of concept, role, and individual names S, the *restriction of a substitution* σ to the elements of S is denoted $\sigma|_S$, given by $\sigma|_S(x) = \sigma(x)$ if $x \in S$, and $\sigma(x) = x$ otherwise.

Let σ be an L-substitution. We recursively define the result of applying an L-substitution σ to the concepts, roles, and axioms in the DL $\mathcal{SROIQ}$ according to Fig. 1. The result of applying σ to an L-knowledge base K is the set of axioms $K\sigma$ obtained by applying σ to every axiom in K.

Closure under substitutions. Let L, K, and σ be as above. Since there may be restrictions on what set of (L-)axioms form L-knowledge bases, $K\sigma$ may not be an L-knowledge base, even though K is. We say that $K\sigma$ is a *legal instance* for L if $K\sigma$ is an L-knowledge base. When L is understood from context, we will simply say that $K\sigma$ is legal. We say that L is closed under substitutions if applying an L-substitution to an L-knowledge base always produces a legal instance.

Working with substitutions and notions based on substitutions in a description logic which is not closed under substitutions is necessarily somewhat more intricate. In particular, the notions of containment presented in Sec. 4 can behave counterintuitively in this case. In this paper we therefore restrict ourselves to fragments of $\mathcal{SROIQ}$ that are closed under substitutions. The general case will be presented in future work.

$\mathcal{SROIQ}$ itself is not closed under substitutions. However, any set of $\mathcal{SROIQ}$ axioms that do not mention role chains is a legal $\mathcal{SROIQ}$ knowledge base (see [16]). Accordingly, $\mathcal{ALCHOIQ}$, the fragment of $\mathcal{SROIQ}$ that does not allow role chains, is closed under substitutions. $\mathcal{SHIQ}$ [15] and $\mathcal{EL}$++ [3] are not closed under substitutions, while $\mathcal{ALC}$ [27] is.

3 Ontology Templates

Let L be a DL language. An L-template is an L-knowledge base T together with a set $\mathsf{param}(T)$ of distinguished concept names, role names, and constants from T called the *parameters*. Given an L-substitution σ, we define the *instance* $T\sigma$ of T to be $T\sigma|_{\mathsf{param}(T)}$.

Example 2. We write $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}) \mapsto \{\mathsf{whole} \sqsubseteq \exists \mathit{hasPart}.\mathsf{part}\}$ to designate the template PartOf which has a single axiom knowledge base $\{\mathsf{whole} \sqsubseteq \exists \mathit{hasPart}.\mathsf{part}\}$ where *hasPart* is a role name and the concept names part and whole are parameters. Let σ be the substitution $\{\mathsf{part} \mapsto \mathit{SteeringWheel}, \mathsf{whole} \mapsto \mathit{Car}\}$, then $\mathsf{PartOf}\sigma = \mathsf{PartOf}(\mathit{SteeringWheel}, \mathit{Car})$ is the following instance of PartOf: $\{\mathit{Car} \sqsubseteq \exists \mathit{hasPart}.\mathit{SteeringWheel}\}$. Note that also $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole})$ is an instance of PartOf with the identity function as substitution; this instance is $\{\mathsf{whole} \sqsubseteq \exists \mathit{hasPart}.\mathsf{part}\}$.

If several templates have been defined, it would be convenient to re-use existing templates to create new ones. To that end, we define a complex template as a template that, in addition to axioms, contains one or more (complex) template instance statements. To avoid circular templates that cannot meaningfully be expanded, we demand that the digraph given by one complex template containing an instance statement of another template be acyclic. In addition, to ensure unique expansions we pose requirements essentially to the effect that parameters in a complex template do not occur as non-parameters in its descendants. Instantiating a complex template T using a substitution σ is then defined as before, with the application of σ to an instance statement $T\tau$ being $T\tau\sigma$. It is clear that a complex template can be re-written into an equivalent non-complex template, by recursively replacing each template instance statement with the corresponding instance. Doing so is called *expanding* a complex template. For the remainder of the paper, we assume that all templates are expanded, i.e., non-complex, unless noted otherwise.

Example 3. Let $\mathsf{QualityValue}$ be the template

$$\mathsf{QualityValue}(\mathsf{x}, \mathsf{hasQuality}, \mathsf{uom}, \mathsf{val}) \mapsto$$
$$\{\mathsf{x} \sqsubseteq \exists \mathsf{hasQuality}.(\forall \mathit{hasDatum}.(\exists \mathit{hasUOM}.\{\mathsf{uom}\} \sqcap \exists \mathit{hasValue}.\{\mathsf{val}\}))\}$$

which intuitively expresses that x has a quality with a given value val with a given unit of measure uom. Using this template and the PartOf template from Ex. 2, we can define the complex template $\mathsf{PartLength}$ as

$$\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length}) \mapsto \{\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}),$$
$$\mathsf{QualityValue}(\mathsf{part}, \mathit{hasLength}, \mathit{meter}, \mathsf{length})\}$$

which expresses that the whole has a part with a given length measured in meters. An example instance of the template is

$$\mathsf{PartLength}(\mathit{Porsche123}, \mathit{SoftTop}, \mathit{3.40})$$

stating that (the car) Porsche123 has a softtop (roof) of length 3.40 meters. The expansion of $\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length})$ is

$$\begin{aligned}\{&\mathsf{whole} \sqsubseteq \exists \mathit{hasPart}.\mathsf{part},\\ &\mathsf{part} \sqsubseteq \exists \mathit{hasLength}.(\forall \mathit{hasDatum}.(\exists \mathit{hasUOM}.\{\mathit{meter}\}\\ &\qquad\qquad\qquad\qquad\qquad\qquad \sqcap \exists \mathit{hasValue}.\{\mathsf{length}\}))\}.\end{aligned}$$

A clear motivation for using templates as ontology macros is as a practical implementation for applying and composing ontology patterns when engineering ontologies, as mentioned in Sec. 1. Currently, ODPs offer little framework for actually using the patterns in the construction of an ontology other than importing and/or copying and extending the ontology representation of the pattern manually. There is no prescribed way of representing if and how a pattern has been used.

By regarding an ODP as a template where the relevant concepts, roles and individuals are marked as parameters, we can simply instantiate the pattern to get a customised copy of it fit for the ontology engineering task at hand. The template instance expression serves as documentation of how the template pattern is used. Creating new patterns using the techniques in [12,26], e.g., specialising, cloning, composing and templating, is made simple with templates, as illustrated by the examples above. A set of ODPs represented as templates can actively be used in the ontology engineering process as an abstraction interface for representing ontological statements at a suitable level of granularity, i.e., possibly in their unexpanded form. The templates themselves may be constructed and maintained, both in isolation and as an expanded ontology, using available ontology editors and reasoners.

3.1 Templates as Queries

A template T can also be viewed as a query, retrieving from some knowledge base O all concept and role expressions as well as individuals that, when used as input to T, would produce a subset of O — a notion dual to that of a template instance.

Formally, the result of evaluating T on O is the set of substitutions $\mathsf{eval}(T, O) = \{\sigma|_{\mathsf{param}(T)} \mid T\sigma$ a satisfiable instance such that $T\sigma \subseteq O\}$. We require satisfiability to make templates as queries well-behaved even in the case of an unsatisfiable KB O. This becomes particularly useful when O is unsatisfiable due to axioms that have nothing to do with T.

Example 4. Let $\mathsf{PartLength}$ be the template as defined in Ex. 3. To get all lengths of the parts of a Porsche123, we can use the following template as query: $\mathsf{PartLength}(\mathit{Porsche123}, \mathsf{part}, \mathsf{length})$. Evaluating this template over an ontology containing the example instance in Ex. 3 will give an answer set containing the single substitution $\{\mathsf{part} \mapsto \mathit{SoftTop}, \mathsf{length} \mapsto \mathit{3.40}\}$.

Using templates as queries for extracting pattern instances may be useful in many cases; for instance, representing an ontology as a set of template instances, rather than its DL axioms, should convey a better picture of the contents of the ontology to a human user. Also, there is a case to made for using templates both as macros and as queries in data exchange settings and for translating between different ontology languages. This is discussed further in Sec. 5. When querying an ontology O with a template T one might be interested in answers which are, e.g., semantically entailed by O, rather than syntactically stated in O. We point to this issue again in Sec. 6. For our current, expository, purposes, however, the simpler syntactic definition of $\mathsf{eval}(T, O)$ is sufficient.

Complexity. Deciding whether there exists a substitution such that $T\sigma \subseteq O$ is NP-complete. Membership is obvious, while for hardness, we reduce from 3-colourability by using axioms of the form $A \sqcap B \sqsubseteq \top$ to simulate the edges of the input graph, where A, B are vertices. Let this KB be the template T, with every vertex a parameter. We likewise encode K_3, the complete graph on three vertices, and call it O. It is clear that the input graph is 3-colourable if and only if there exists a substitution σ such that $T\sigma \subseteq O$.

Therefore, the complexity of deciding whether $\mathsf{eval}(T, O)$ is empty is the greater of NP and the complexity of checking whether a candidate substitution σ is such that $T\sigma$ is a satisfiable instance. The complexity of checking satisfiability depends on the language of T, and is a well-studied topic; see [8] for an overview.

4 Reasoning about Templates

Whether viewing templates as macros or queries, it would be useful to be able to discuss and decide various relationships between them. For templates, a natural idea is to consider notions of containment between the instances produced by two templates. Below, we define two such notions, one based on set inclusion and corresponding to containment of templates as queries, and one based on logical consequence. For the latter, we will show in Sec. 5 how it can be applied to the setting of data exchange.

Furthermore, we consider the question of whether a template T can be used to describe a given knowledge base. Being able to decide this is important when considering the practical usability of a set of templates, and show a simple characterisation for it.

In the following, we assume that T_1 and T_2 are L-templates such that $\mathsf{param}(T_1) \subseteq \mathsf{param}(T_2)$. We say that T_1 is *syntactically contained* in T_2, written $T_1 \subseteq_s T_2$, if $T_1\sigma \subseteq T_2\sigma$ for every substitution σ. It is worth noting that the the requirement $\mathsf{param}(T_1) \subseteq \mathsf{param}(T_2)$ is only seemingly restrictive to this definition: in fact, syntactic containment implies parameter containment.

The following theorem characterises the relationship between syntactic containment and the template parameters.

Theorem 1. *Let T_1 and T_2 be L-templates. Then the following are equivalent:*

1. $T_1 \subseteq_s T_2$;
2. $T_1 \subseteq T_2$ *and no axiom or assertion in* T_1 *contains a parameter from* $\mathsf{param}(T_2) \setminus \mathsf{param}(T_1)$
3. $T_1\sigma' \subseteq T_2\sigma'$, *where* σ' *maps each concept, role, or individual name to a new, previously unused name.*

Proof. $1 \rightarrow 2$: Choosing σ as the identity shows that $T_1 \subseteq T_2$. Let $\varphi(P_2)$ be an axiom or assertion in T_1 such that $P_2 \in \mathsf{param}(T_2) \setminus \mathsf{param}(T_1)$. Then $\varphi\sigma|_{T_1}(P_2) \in T_1\sigma$ for every σ. Choose σ' such that $\sigma'(P_2) = P_2'$ for a new and previously unused symbols P_2' and the identity otherwise. Applying $\sigma'|_{T_1}$ to $\varphi(P_2)$ yields $\varphi\sigma'|_{T_1}(P_2)$, while σ'_{T_2} yields $\varphi\sigma'|_{T_2}(P_2')$. In particular, $\varphi\sigma'|_{T_1}(P_2) \notin T_2\sigma'$ and therefore $T_1\sigma' \nsubseteq T_2\sigma'$.

$2 \rightarrow 3$: Let σ' be as described in the theorem and let $\varphi \in T_1$ be an axiom or assertion. Applying $\sigma'|_{T_1}$ to φ yields the same as applying

$\sigma'|_{T_2}$ to φ: if φ does not contain parameters, then $\varphi\sigma' = \varphi$ in both templates; if it contain parameters they must come from $\mathsf{param}(T_1)$, and hence $\varphi\sigma'|_{T_1} = \varphi\sigma'|_{T_2}$. Hence $T_1\sigma' \subseteq T_2\sigma'$.

$3 \to 1$: Proof by contraposition: Assume σ is a substitution such that $\varphi\sigma|_{T_1}$ is not in $T_2\sigma$ for some axiom or assertion $\varphi \in T_1$. If $\varphi \notin T_2$ then $T_1\sigma' \not\subseteq T_2\sigma'$ for σ' as defined in the theorem. If $\varphi \in T_2$ then φ must contain a parameter P from $\mathsf{param}(T_2) \setminus \mathsf{param}(T_1)$, since $\varphi\sigma|_{T_1} \notin T_2\sigma$. Thus $\varphi\sigma'|_{T_1}$ still contains P, since $\sigma'|_{T_1}$ acts as the identity on names not in $\mathsf{param}(T_1)$. P is not in the image of $\sigma'|_{T_2}$ and therefore $\varphi\sigma'|_{T_1} \notin T_2\sigma'$, i.e., $T_1\sigma' \not\subseteq T_2\sigma'$.

Similarly to above, we say that a template T_1 is *entailment contained* in T_2, written $T_1 \subseteq_e T_2$, if $T_2\sigma \models T_1\sigma$ for every substitution σ.

Theorem 2. *Let T_1 and T_2 be L-templates where $\mathsf{param}(T_1) \subseteq \mathsf{param}(T_2)$. Then $T_1 \subseteq_e T_2$ if and only if $T_2\sigma' \models T_1\sigma'$, where σ' is a substitution that maps each concept, role, and individual name to a fresh, previously unused name.*

Proof. We need only prove the "if" direction, which we show by contradiction. Assume there exists a substitution σ and a model M such that $M \models T_2\sigma$ and $M \not\models T_1\sigma$. We expand this model to M' by taking into account the fresh names from σ': define $M'(P\sigma') = M(P\sigma)$ for parameters $P \in \mathsf{param}(T_2)$. Then by the standard inductive constructions, we get $M'(\varphi\sigma'|_{\mathsf{param}(T_2)}) = M(\varphi\sigma|_{\mathsf{param}(T_2)})$ and hence $M' \models T_2\sigma'$ and $M' \not\models T_1\sigma'$.

Since templates as macros and templates as queries are dual to each other, the fundamental notion of feeding the answers of one query into another (used in many applications for first-order queries) becomes more powerful. In the following, we define this operation formally and give examples of its use.

First, we will need a convenient bit of notation. Given a set of substitutions S, define $\mathsf{inst}_{\mathsf{sat}}(T, S) = \bigcup\{T\sigma \mid T\sigma \text{ satisfiable}, \sigma \in S\}$. Then a template T is said to *produce* a knowledge base O if there exists a set of substitutions S such that $O = \mathsf{inst}_{\mathsf{sat}}(T, S)$. Now, we define the move or materialise operator as follows: Let T_1 and T_2 be two L-templates with $\mathsf{param}(T_1) \subseteq \mathsf{param}(T_2)$, and let O be a knowledge base. We define $M(O, T_1, T_2) = \mathsf{inst}_{\mathsf{sat}}(T_2, \mathsf{eval}(T_1, O))$. This operator can be used to decide whether a knowledge base O is comprised entirely of instances of a given template T_1, by checking whether $M(O, T_1, T_1) = O$. The axioms in $O \setminus M(O, T_1, T_1)$, if not empty, are the ones that do not match any axioms in the template. This is summarised in the following theorem.

Theorem 3. *Let T be a template and O a knowledge base. T can produce O if and only if $M(O, T, T) = O$.*

5 Using Templates for Data Exchange

Templates can be used for data exchange. In particular, observe that two L-templates T_S and T_T with $\mathsf{param}(T_T) = \mathsf{param}(T_S)$ define an implicit mapping from any satisfiable instance $T_S\sigma$ to the instance $T_T\sigma$. This observation can be exploited for data exchange.

Let O_S be a source knowledge base, and T_S a template describing the concepts, roles, and individuals from O_S that we wish to transfer to a different setting (e.g., into some other knowledge base). To do so, we use a target template T_T, potentially by renaming parameters in an existing template that we have used to build other parts of our target KB. We can now use T_S and T_T as the body and head of a data exchange mapping. The result of applying such a mapping is precisely captured by the operator M defined in Sec. 4.

Example 5. Assume that the knowledge base O_C models partonomy relationships (between car components) using the role *isComponentOf*, and suppose we want to translate these relations into using the *hasPart* role used by the PartOf template as defined in Ex. 2. This can be achieved by the following data exchange setting using the PartOf template as query and the following template as macro: $\mathsf{CMPT}(\mathsf{part}, \mathsf{whole}) \mapsto \{\mathsf{part} \sqsubseteq \exists \mathit{isComponentOf}.\mathsf{whole}\}$. The materialisation of the exchange is

$$M(O_C, \mathsf{PartOf}, \mathsf{CMPT}) = \mathsf{inst}_{\mathsf{sat}}(\mathsf{CMPT}, \mathsf{eval}(\mathsf{PartOf}, O_C)).$$

Let $O_T = M(O_S, T_S, T_T)$. It is possible that T_T is more restrictive than T_S when it comes to satisfiable instances. If so, then some of the substitutions retrieved by T_S create unsatisfiable instances of T_T, which are then discarded. We can check whether this occurs by reversing the materialisation, and considering the KB $O'_S = M(O_T, T_T, T_S)$. Since O_S may have a lot of statements not retrieved by T_S in the first place, the two are not comparable. However, consider the subset of O_S that T_S does retrieve, that is, $M(O_S, T_S, T_S) \subseteq O_S$.

It is always the case that $M(O_T, T_T, T_S) \subseteq M(O_S, T_S, T_S)$, since for every axiom ϕ in $M(O_T, T_T, T_S)$ we have that there exists a substitution σ such that $\phi \in T_S\sigma$. Then, we have that $T_T\sigma \subseteq O_T = M(O_S, T_S, T_T)$, and hence that $T_S\sigma \subseteq O_S$. Therefore, $\sigma \in \mathsf{eval}(O_S, T_S)$, and hence $\phi \in M(O_S, T_S, T_S)$.

If this inclusion is an equality, then in terms of concepts retrieved and inserted, we have lost no information. In fact, we have kept the syntactic form of the axioms as they appear in the templates. When is this always the case, i.e., $M(O_T, T_T, T_S) = M(O_S, T_S, T_S)$ for every O_S? Recall that we assumed that L is closed under substitutions — it follows that this becomes a question of conditional satisfiability. We want the following property: For every σ, if $T_S\sigma$ is satisfiable, then so is $T_T\sigma$. If the DL language we are working in allows unrestricted equivalence axioms, then this problem is equivalent to checking whether there exists a set of equivalence axioms $S = \{A \equiv \phi \mid A \in \mathsf{param}(T_S), \phi$ a concept/role expression or individual name$\}$ such that $T_S \cup S$ is satisfiable but $T_T \cup S$ is not. This is a difficult problem, but as a preliminary result, we can show that entailment containment is sufficient, and hence that syntactic containment is also.

Theorem 4. *Let L be a DL language, and T_S, T_T be L-templates with $\mathsf{param}(T_S) = \mathsf{param}(T_T)$. We have that $M(O_T, T_T, T_S) = M(O_S, T_S, T_S)$ for every L-KB O_S if $T_T \subseteq_e T_S$. The converse is false.*

Proof. Let $T_S(A) = \{A \sqsubseteq B\}$ and $T_T(A) = \{A \sqsubseteq C\}$. It is clear that for any O_S, all substitutions for A create satisfiable instances of both T_S and T_T. Equally, it is clear that there is no containment between T_S and T_T. On the other hand, let O_S be arbitrary. If $T_T \subseteq_e T_S$, then for every $\sigma \in \mathsf{eval}(O_S, T_S)$, $T_T\sigma$ is satisfiable, and hence $T_T\sigma \subseteq O_T$. Then, we have that $\sigma \in \mathsf{eval}(T_T, O_T)$, and thus that $T_S\sigma \subseteq M(O_S, T_S, T_S)$.

The above notion of not losing information is rather restrictive. More natural is the semantic notion, whether $M(O_T, T_T, T_S) \models M(O_S, T_S, T_S)$. Since equality implies entailment, the above result applies also to this case. However, we do not yet have a characterisation for the general case.

6 Conclusion and Future Work

We have presented ontology templates, a notion of ontology macros for the DL $\mathcal{SROIQ}$. We have shown how such macros can be used to assist knowledge base creation and maintenance, in particular by treating templates as queries as needed. We have also presented preliminary results on relationships between templates, and argued that they have a use within data exchange. In addition to extending our preliminary results, we plan to investigate the areas below.

Closure under substitution. As specified in Sec. 2, we have restricted ourselves to description logics that are closed under substitutions. Several notable description logics, including $\mathcal{SROIQ}$ itself, do not have this property. Thus extending our scope to such DLs is an important current and future consideration. In particular, we are investigating suitable conditions on substitutions that guarantee that an instance of an L-template is always an L-ontology for relevant descriptions logics L. We hope that such conditions will allow us to compare templates across DL languages. For the rest of this section we continue to assume that L is closed under substitution, that all substitutions are L-substitutions, and that all templates T and KBs O are in L. However, the research questions below have natural extensions to description logics that are not closed under substitution.

Queries and entailment relations. Recall from Sec. 3.1 that a template T can be used as a query on a KB O. Intuitively, such a query searches O for instances of the pattern expressed by T. According to the definition in Sec. 3.1, the query returns the parameter substitutions σ such that $T\sigma \subseteq O$ and $T\sigma$ is satisfiable.

Since this is formulated in terms of set inclusion, evaluating T over O returns only the instances of T that are explicitly in O. However, one might also be interested in instances that are entailed by O, for whatever notion of entailment one finds relevant. For semantic entailment, for instance, the results of the query would be parameter substitutions σ such that $T\sigma$ is satisfiable and $O \vDash T\sigma$.

Example 6. Let T be the template $\mathsf{P}(\mathsf{x},\mathsf{y}) \mapsto \{\mathsf{x} \sqsubseteq \mathsf{y}\}$ and O the KB $\{A \sqsubseteq B,\ B \sqsubseteq C\}$. The result of querying O with T with respect to set inclusion and semantic entailment, respectively, are $\mathsf{eval}(T,O) = \{\{x \mapsto A, y \mapsto B\}, \{x \mapsto B, y \mapsto C\}\}$ and $\mathsf{eval}_{\vDash}(T,O) = \{\{x \mapsto \phi, y \mapsto \psi\} \mid \phi, \psi$ L-concept expressions, and $O \vDash \phi \sqsubseteq \psi\}$.

In particular, $\{x \mapsto A, y \mapsto C\}$ is not in $\mathsf{eval}(T, O)$, but it is in $\mathsf{eval}_{\models}(T, O)$.

In general, $\mathsf{eval}_{\models}(T, O)$ is an infinite set, containing "redundant" concept and role expressions such as $A \sqcap A$, as well as concept names that do not occur in O. Thus on the one hand, $\mathsf{eval}(T, O)$ is too restrictive. On the other hand, $\mathsf{eval}_{\models}(T, O)$ is too permissive, resulting in an infinite set on the trivial ontology of Ex. 6, and allowing clearly redundant answers. We plan to investigate notions of evaluation of a template over an ontology that lie between these two extremes.

Knowledge base translations and templates. Let $\vdash$ be an entailment relation between L-ontologies, such as set inclusion or semantic entailment. Together with the notion of L-substitution, this induces a mapping, or translation, between ontologies. A translation $f : O \to O'$ is a substitution σ with the properties that 1) for any name x, if $\sigma(x) \neq x$ then x occurs in O and $f(x)$ is an expression constructed from names occurring in O', and 2) $O' \vdash O\sigma$. Such translations compose, and so form a category, i.e. a formal system of morphisms. Although well-known, such translations do not seem to have been studied in the DL context.

Templates lend themselves well to be studied in the setting of formal systems of ontologies and translations. First, they can be defined there, as simply a special kind of translation between two ontologies. Second, the operations involving templates, e.g. extending an ontology by a template instance and querying an ontology with a template, can be represented using basic categorical operations.

In general, the study of templates in the system of ontology translations provides a flexible and formal language and framework for the representation of templates and the formulation and study of their relations. Further operations that naturally suggest themselves in this setting are, in particular, allowing for the parameters themselves to form an ontology, different from the template ontology. This allows for conditional extensions of ontologies by template instances; the template first queries the ontology for an instance of the parameter pattern, and if one is found, it extends this pattern with an instance of the template pattern. It also becomes possible to use templates as a form of constraint, expressing that whenever an instance of the parameter pattern is present in an ontology, the ontology must also witness the corresponding instance of the template pattern.

References

1. Ontology templates. `http://swtmp.gitlab.io`.
2. M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In J. Paredaens and D. V. Gucht, editors, *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'10)*, pages 227–238. ACM, 2010.
3. F. Baader, S. Brandt, and C. Lutz. Pushing the EL envelope. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 364–369. Professional Book Center, 2005.
4. M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and mmsnp. In *Proceedings of the 32Nd Symposium on Principles of Database Systems*, PODS '13, pages 213–224, New York, NY, USA, 2013. ACM.
5. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
6. S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language. Technical report, W3C, 2012.
7. G. De Giacomo, M. Lenzerini, and R. Rosati. Higher-order description logics for domain metamodeling. In W. Burgard and D. Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.
8. F. M. Donini. Complexity of reasoning. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 96–136. Cambridge University Press, 2003.
9. A. Gangemi and V. Presutti. *Ontology Design Patterns*, pages 221–243. Springer, 2009.
10. K. Hammar. Ontology Design Patterns in WebProtege. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, 2015.
11. K. Hammar et al. *Collected Research Questions Concerning Ontology Design Patterns*, chapter 9, pages 189–198. Volume 025 of Hitzler et al. [14], 2016.
12. K. Hammar and V. Presutti. Template-Based Content ODP Instantiation. Workshop on Ontology and Semantic Web Patterns, WOP 2016.
13. L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi. A.: Rdf123: from spreadsheets to rdf. In *ISWC*. Springer, 2008.
14. P. Hitzler et al., editors. *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, volume 025. IOS Press, Amsterdam, 2016.
15. I. Horrocks. Practical reasoning for very expressive description logics. In *Journal of the Interest Group in Pure and Applied Logics 8*, pages 293–323, 2000.
16. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *KR*, pages 57–67. AAAI Press, 2006.
17. L. Iannone, A. L. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009.
18. S. Jupp et al. Populous: a tool for building OWL ontologies from templates. *BMC Bioinformatics*, 13(S-1):S5, 2012.
19. J. W. Klüwer, M. G. Skjæveland, and M. Valen-Sendstad. ISO 15926 templates and the Semantic Web. W3C Workshop on Semantic Web in Oil & Gas Industry, 2008.
20. A. Langegger. Xlwrap — querying and integrating arbitrary spreadsheets with sparql. In *ISWC*, pages 359–374, 2009.

21. D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Mapping analysis in ontology-based data access: Algorithms and complexity. In *International Semantic Web Conference (1)*, volume 9366 of *Lecture Notes in Computer Science*, pages 217–234. Springer, 2015.
22. P. Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. In *OWLED*, 2013.
23. B. Motik. On the properties of metamodeling in OWL. *J. Log. Comput.*, 17(4):617–637, 2007.
24. M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. M2: A Language for Mapping Spreadsheets to OWL. In *OWLED*, 2010.
25. C. Ogbuji. Infixowl: An idiomatic interface for owl. In *OWLED*, 2008.
26. V. Presutti and A. Gangemi. Content Ontology Design Patterns As Practical Building Blocks for Web Ontologies. In *ER*, pages 128–141. Springer, 2008.
27. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1 – 26, 1991.
28. V. Vassiliadis, J. Wielemaker, and C. Mungall. Processing owl2 ontologies using thea: An application of logic programming. In *OWLED*, pages 89–98. CEUR-WS.org, 2009.
29. D. Vrandečić. Explicit knowledge engineering patterns with macros. In *Proceedings of the Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, 2005.
30. Z. Xiang, J. Zheng, Y. Lin, and Y. He. Ontorat: automatic generation of new ontology terms, annotations, and axioms based on ontology design patterns. *Journal of Biomedical Semantics*, 6(1):4, 2015.

Paper 3

Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates

Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler. “Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates.” In: *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017.* 2017 [P3]

Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates

Martin G. Skjæveland[1], Henrik Forssell[1], Johan W. Klüwer[2], Daniel Lupp[1], Evgenij Thorstensen[1], and Arild Waaler[1]

[1] Department of Informatics, University of Oslo
{martige,jonf,danielup,evgenit,arild}@ifi.uio.no
[2] DNV GL, Norway
johan.wilhelm.kluewer@dnvgl.com

Abstract. Reasonable Ontology Templates, OTTRs for short, are OWL ontology macros capable of representing ontology design patterns (ODPs) and closely integrating their use into ontology engineering. An OTTR is itself an OWL ontology or RDF graph, annotated with a special purpose OWL vocabulary. This allows OTTRs to be edited, debugged, published, identified, instantiated, combined, used as queries and bulk transformations, and maintained—all leveraging existing W3C standards, best practices and tools. We show how such templates can drive a technical framework and tools for a practical, efficient and transparent use of ODPs in ontology design and instantiation. The framework allows for a clear separation of the design of an ontology, typically managed by ontology experts, and its bulk content, provided by domain experts. We illustrate the approach by reconstructing the published Chess Game ODP and producing linked chess data.

1 Introduction

Ontology-based methods have matured to where they offer knowledge workers practical solutions for data management. In particular, tools that support W3C recommendations, such as reasoning tools for OWL ontologies, are sufficiently stable and efficient to allow wide-scale industrial use. However, from the perspective of product vendors and consultancy companies in the IT industry, ontologies are still viewed as a fringe technology. Hence ontology-based solutions are rarely proposed to enterprise customers, and support from the software industry is quite limited. One factor that impedes uptake is the high cost of establishing and maintaining a high-quality ontology. In part this is due to the scarcity of ontology experts, with availability in most cases below critical mass.

Ontology design patterns (ODPs) [9] serve the purpose of alleviating some of the difficulties involved with creating ontologies by offering reusable, best-practice building blocks and structures for ontology construction, commonly implemented and published as small OWL ontologies. Methods for combining and instantiating ODPs are described [20,8], and a methodology for building ontologies using patterns exists [1]. However, while ODPs are often presented as "practical building blocks" [20], we argue that ODPs in their current form, i.e., as found on `http://ontologydesignpatterns.org/` featuring a graphical representation, a description and a "reusable OWL building block", are not practical enough, especially for the development of large ontologies—as using and adapting ODPs to a particular modelling task currently often require considerable manual work. What is needed are better tool supported methods for applying ODPs in ontology engineering. Efficient tool support is imperative to industrial scale deployment of ontology-based methods.

The work reported on in this paper has the potential to remedy the situation; *Reasonable Ontology Templates* (OTTRs) are simple, but powerful, templates or macros for ontologies, cf. [21], represented in OWL using a dedicated OWL vocabulary. An OTTR can be viewed as a *parameterised ontology* which can be nested, i.e., defined using other OTTRs, and *instantiated* by providing arguments to fit the parameters of the template. By recursively *expanding* an OTTR by replacing any containing OTTR with the pattern it represents, we obtain a regular OWL ontology. Using this feature, we can reason both on the OTTR specification and its expansion, and additionally leverage existing W3C languages and tools for different ontology engineering tasks—all driven by OTTRs. Specifically, the implicit mapping between an OTTR's parameters and its pattern may be exploited to *generate* various format descriptions and transformation specifications, e.g., queries for extracting pattern instances and transformations between tabular input formats and OTTR pattern instances that may be processed by readily available desktop tools. The only additional tool support that is needed to make use of OTTRs, are tools that can perform the relatively simple operation of template expansion and instantiation.

In addition to supporting the work of the ontology engineer, we believe OTTRs can provide a framework whereby a few ontology experts can serve a large number of domain experts and put these in position to actively contribute to the development and maintenance of ontologies. This is achieved by clearly separating the design of an ontology and its bulk content: The ontology expert designs and combines patterns represented as OTTRs to provide *user-facing* patterns on a level of abstraction suitable for the domain experts. From the user-facing OTTRs a simple input format is generated together with a transformation specification of the input format to ontology format. The task of the domain expert is then “only” to provide instance arguments to the input format.

Sec. 2 defines OTTR templates and introduces the OTTR OWL vocabulary for expressing them for use on the semantic web. Furthermore, we explain how OTTRs may be used for driving a technical framework for different ontology engineering tasks, and also give a prototype implementation that can serve the various specifications for such a framework. Sec. 3 discusses the particular application of OTTRs for using ODPs for ontology design and instantiation, illustrated on the Chess Game ODP, and for linked data publication. In Sec. 4 we discuss the benefits and shortcomings for OTTRs and compare with related work. We conclude with Sec. 5.

2 Reasonable Ontology Templates

In the following we introduce the core concepts *template*, *template instance* and *expansion* through examples and by alluding to basic description logic concepts; see [6] for a more formal and thorough description.

A *template* $\mathcal{T}$ is a knowledge base $\mathcal{O}_\mathcal{T}$ and a list of parameters $\mathrm{param}(\mathcal{T}) = (\mathsf{p}_1, \ldots, \mathsf{p}_\mathsf{n})$ of distinguished concept, role, or individual names from $\mathcal{O}_\mathcal{T}$. We write a template as

$$\mathcal{T}(\mathsf{p}_1, \ldots, \mathsf{p}_\mathsf{n}) :: \mathcal{O}_\mathcal{T}.$$

and refer to the left side as the *head* and the right side as the *body*. For a list of parameters $(q_1, \ldots, q_n)$ we call $\mathcal{T}(q_1, \ldots, q_n)$ a *template instance*. Intuitively, a template instance is shorthand for representing a specific occurrence or instance of a pattern. More precisely, the *expansion* of $\mathcal{T}(q_1, \ldots, q_n)$ is the ontology $\mathcal{O}_\mathcal{T}(q_1, \ldots, q_n)$ obtained by replacing each parameter occurrence of p_i in $\mathcal{O}_\mathcal{T}$ with the argument q_i, for all $1 \leq i \leq n$.

Example 1. $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}) :: \{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$ is the template PartOf which has a single axiom knowledge base $\{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$ where $\mathrm{hasPart}$ is a role name and part and whole are parameters. $\mathsf{PartOf}(\mathrm{SteeringWheel}, \mathrm{Car})$ is an instance of PartOf representing the ontology $\{\mathrm{Car} \sqsubseteq \exists \mathrm{hasPart}.\mathrm{SteeringWheel}\}$. Note that also $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole})$ is an instance of PartOf, where the parameter names are substituted for themselves; its ontology is $\{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$.

In addition to ontology axioms, a template may also contain template instances in its body. The notion of template instance expansion is then extended to a recursive operation where any template instances in the template body are expanded tail-recursively. Cyclic template definitions are not allowed.

Example 2. Let $\mathsf{QualityValue}$ be the template

$$\mathsf{QualityValue}(\mathsf{x}, \mathsf{hasQuality}, \mathsf{uom}, \mathsf{val}) :: \\ \{\mathsf{x} \sqsubseteq \exists\mathsf{hasQuality}.(\forall\mathrm{hasDatum}.(\exists\mathrm{hasUOM}.\{\mathsf{uom}\} \sqcap \exists\mathrm{hasValue}.\{\mathsf{val}\}))\}$$

which intuitively expresses that x has a quality with a given value val with a given unit of measurement uom. Using this template and the PartOf template from Ex. 1, and fixing some of the parameters, the template $\mathsf{PartLength}$ is defined as

$$\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length}) :: \{\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}), \\ \mathsf{QualityValue}(\mathsf{part}, \mathrm{hasLength}, \mathrm{meter}, \mathsf{length})\}$$

which expresses that the whole has a part with a given length measured in meters. An example instance of the template is $\mathsf{PartLength}(\mathrm{2CV}, \mathrm{SoftTop}, 1.40)$ stating that (the car) 2CV has a softtop (roof) of length 1.40 meters. The expansion of the instance $\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length})$ is

$$\{\mathrm{2CV} \sqsubseteq \exists\mathrm{hasPart}.\mathrm{SoftTop}, \\ \mathrm{SoftTop} \sqsubseteq \exists\mathrm{hasLength}.(\forall\mathrm{hasDatum}. \\ (\exists\mathrm{hasUOM}.\{\mathrm{meter}\} \sqcap \exists\mathrm{hasValue}.\{1.40\}))\}.$$

2.1 The OTTR OWL vocabulary

We adapt ontology templates to the semantic web by serialising them using RDF with the OTTR[3] OWL vocabulary defined specifically for this task. A template is associated with an RDF graph [3] (document) available at the IRI of the template.[4] The RDF graph contains both the head, identifying the template and its parameters, and the body of the template. The template body may contain template instances and other ontology axioms, which

[3] We use OTTR to designate the vocabulary. All other unprefixed, inline typewriter font words refer to OTTR vocabulary elements.

[4] Similarly as for OWL ontologies [17, section 3.2 Ontology Documents]

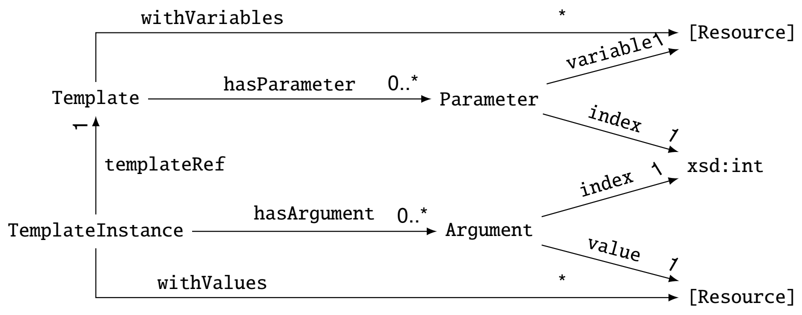

Fig. 1: High-level overview of the OTTR OWL vocabulary.

are expressed using regular RDF/OWL serialisation [19]. (Note that the RDF graph need not represent an OWL ontology. In fact, templates may be used in a more generic way as "RDF macros". However, we prefer to call them OWL macros in order to clearly indicate their applicability to ontology engineering, reasoning and ontology design patterns.) Parameters and arguments are represented as named variables and named values, respectively, where the name is represented in RDF as an IRI, and variables and values may be any RDF resource, i.e., any IRI or literal [3].

The most prominent elements of the OTTR vocabulary are the following, see also Fig. 1 for a graphical overview. A `Template` has zero or more `Parameters`. Each `Parameter` is consecutively numbered by an integer valued `index`, starting at 1, and has a `variable` which represents the parameter in the template body. The existence of a `Template` in an RDF graph declares the graph as a template, and an RDF graph specifying a template must contain only one `Template` object. A `TemplateInstance` must refer to a single `Template` via a `templateReference`, and have `Arguments` to match the `Parameters` of the `Template`. An `Argument` must have a `value` and refer a `Parameter` by using the same `index` value as the `Parameter`'s. The range of the `variable` and `value` properties is any RDF resource. These conditions and more are represented in OWL using the OTTR vocabulary available from `http://ns.ottr.xyz/`.

We differentiate between the head and the body of a template represented in RDF using the concept of graph neighbourhood, which informally are all the outgoing triples from the template and parameter individuals.

Definition 1. *Let G be an RDF graph (represented as a set of triples), and r an IRI. We define the* out-neighbourhood of r in G, *denoted* $\mathsf{out}(r, G)$ *as the set of triples $\langle r, x, y\rangle \in G$. Let G be the RDF graph associated with a template with IRI t and parameter IRIs $p_1, \ldots, p_n$. We define the* head *of* *the template in G as* $\mathsf{head}(t) = \bigcup_{x \in \{t, p_1, \ldots, p_n\}} \mathsf{out}(x, G)$, *and the* body *of t in G as* $\mathsf{body}(t) = G \setminus \mathsf{head}(t)$.

A template instance is expanded by copying the template RDF graph[5] to which the instance refers and for each template parameter substituting *all* occurrences of the parameter variable in the RDF graph with its matching argument value. The expansion is applied recursively.

Example 3. The PartOf template from Ex. 1 is represented is the OTTR vocabulary as follows:[6]

```
@prefix ottr:    <http://ns.ottr.xyz/templates#> .
@prefix partOf:  <http://www.ontologydesignpatterns.org/cp/owl/partof.owl#> .
@prefix :        <http://draft.ottr.xyz/i17/partof#> .
  ### head:
<http://draft.ottr.xyz/i17/partof> a owl:Ontology , ottr:Template ;
    owl:imports <http://www.ontologydesignpatterns.org/cp/owl/partof.owl> ;
    ottr:hasParameter [ ottr:index 1; ottr:variable :Whole ] ,
                      [ ottr:index 2; ottr:variable :Part  ] .
  ### body:
:Part  a owl:Class .
:Whole a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
        owl:onProperty partOf:hasPart ; owl:someValuesFrom :Part ] .
```

Observe that the RDF graph is a regular OWL ontology using the OTTR vocabulary to identify the template and its parameters: The template contains a head and body as indicated by the comments, and specifies two parameters with respectively the variables `:Whole` and `:Part`. These variables are used in the template body as regular RDF resources. An instance of this template, reflecting the instance in Ex. 1, is represented as follows.

```
[] ottr:templateRef <http://draft.ottr.xyz/i17/partof> ;
   ottr:hasArgument [ ottr:index 1 ; ottr:value ex:Car ] ,
                    [ ottr:index 2 ; ottr:value ex:SteeringWheel ] .
```

The instance refers to the template with `templateRef`, and each argument refers to a parameter of the template using indices. The expansion of the instance is created by replacing, in a copy of the template RDF graph, all occurrences of `:Whole` with `ex:Car`, and `:Part` with `ex:SteeringWheel`.

[5] Also useful is the expansion procedure that only copies the template body.
[6] Note that all example templates are published at their IRI address.

In order to support a more terse specification of templates and instances, the OTTR vocabulary allows for the use of RDF lists [3] to specify template parameters and instance arguments. Since the RDF list structure is reserved for the serialisation of OWL and therefore not permissible for use in valid OWL2 DL ontologies, the OTTR vocabulary also defines a linked list structure [4], called `List`. Lists may be serialised using either RDF lists or OTTR's `Lists`. The list feature may be used for directly giving the parameter variables of a template, using `withVariables`; and the argument values of an instance, using `withValues`.

Example 4. Using the RDF list format, the template PartLength given in Ex. 2 can be compactly represented:

```
<http://draft.ottr.xyz/i17/partLength> a owl:Ontology , ottr:Template ;
    ottr:withVariables ( :Whole :Part 99 )
[]  ottr:templateRef <http://draft.ottr.xyz/i17/partof> ;
    ottr:withValues ( :Whole :Part ) .
[]  ottr:templateRef <http://draft.ottr.xyz/i17/qualityvalue> ;
    ottr:withValues ( :Part ex:hasLength ex:meter 99 ).
```

Lists may also be used as argument values, supporting patterns which naturally permit variable sized input. Expanding an instance of a template allowing list input, will for each list valued argument replace all occurrences of lists in the template with the same contents as the matching parameter value list.

Example 5. The EquivObjectUnionOf template takes two arguments, a class U and a list of classes, and defines the union of the list of classes as equivalent to U.

```
<http://candidate.ottr.xyz/owl/axiom/EquivObjectUnionOf> a ottr:Template ;
    ottr:withVariables ( :U ( :A :B ) ).
:U rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ; owl:unionOf ( :A :B ) ] .
```

Notice that we here use the list format both to specify the template's two variables, and the second parameter's list variable. An example instance of this template is the following.

```
[]  ottr:templateRef <http://candidate.ottr.xyz/owl/axiom/EquivObjectUnionOf> ;
    ottr:withValues ( ex:Fruit ( ex:Apple ex:Orange ex:Melon ) ).
```

When expanding the instance, all occurrences of lists with the contents `:A :B` in the template will be replaced with a list (copy) containing the elements `ex:Apple ex:Orange ex:Melon`.

In addition to providing a vocabulary for expressing templates, the OTTR ontology includes axioms that allows regular ontology reasoners to check the consistency of OTTR templates, such as domain and range axioms, and functional and key constraints of properties. Also, each parameter variable may be assigned a *type* using different "variable" properties, such as `classVariable`, which informally are subproperties of the `variable` property.[7] The available types reflect the generic classes from the RDF(S) [2] and OWL [19] vocabularies and are arranged in a taxonomy accordingly, where many types are made incompatible using disjoint property ranges; consult the OTTR vocabulary for details. This simple type checking feature is very useful when constructing templates which typically pass on parameters as arguments to other templates, allowing a parameter to be assigned multiple, and possibly incompatible, types. An ontology reasoner will reveal such an inconsistency by reasoning on the OTTR vocabulary of the expanded template. An implementation of the template mechanism should also exploit the type information to check whether instance arguments respect the type of their matching parameter.

Example 6. Assume the PartOf template from Ex. 3 types both parameter variables as classes, using the `classVariable` property, here showing only the head:

```
<http://draft.ottr.xyz/i17/partof> a owl:Ontology , ottr:Template ;
   ottr:hasParameter [ ottr:index 1; ottr:classVariable :Whole ] ,
                     [ ottr:index 2; ottr:classVariable :Part  ] .
```

Now assume the variable of the first parameter of the PartLength template in Ex. 4 is (wrongly) typed as an annotation property using `annotationPropertyVariable`. Then PartLength is inconsistent, since its `:Whole` variable is classified as the disjoint classes `Class` and `AnnotationProperty`.

The OTTR ontology is defined by two ontology documents: *templates-lite* and *templates-core*. The former declares only the vocabulary with domain and range axioms and is useful when the task is primarily to instantiate templates, hence reasoning over template specifications is usually not required. The latter ontology imports the first and enables the reasoning capabilities presented above.

[7] We say "informal" since, in order to support reasoning, the specialisations of the OWL annotation property `variable` are either object properties or datatype properties, and these OWL property types are mutually disjoint.

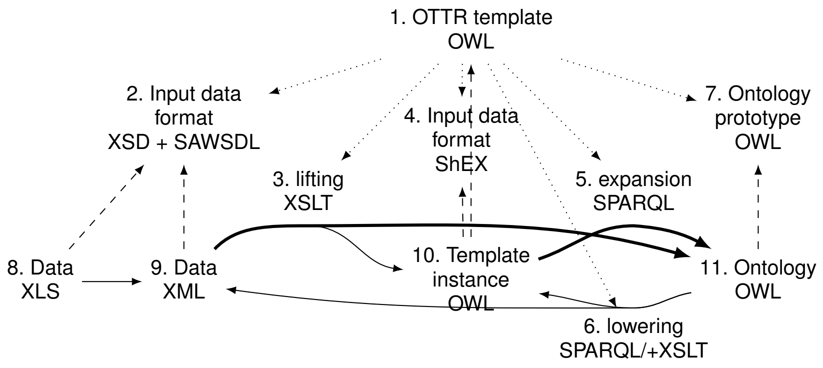

Fig. 2: OTTR-driven ontology construction. The nodes in the diagram each represent a document used in the process. All data format and transformation W3C specifications (2–7) are generated from a template specification, as indicated by the dotted edges. The dashed edges indicate an "instance of" relation, and the solid edges show the flow of the ontology bulk data, highlighting the main routes. The XSD document (2) specifies a "tabular" template instance data input format for XML data (9) that is also supported by some spreadsheet applications (8). The XML data is lifted [5] using XSLT transformations (3) to either the RDF/OWL template instance format (10) which may be validated by a ShEx shape expression (4), or directly to a regular OWL ontology (11). Instances (10) may also be expanded with generated SPARQL queries (5), and be extracted and lowered [5] with SPARQL and XSLT (6) to OWL/RDF and XML format, respectively. The lifting (3) and lowering (6) scripts are identified using SAWSDL in (2).

2.2 Ontology Construction Framework

A core feature of OTTR templates is the ability to relate a simple tabular input format, the template head, to a rich ontological structure in the template body, possibly specified via compositions of other templates. The fact that a template specifies both a tabular input format, an output ontology, and a mapping between the two formats may be exploited by leveraging the capabilities of existing W3C standards and implementations: In addition to specifying an ontology representing a prototypical ontology of the template (the expanded template) in OWL, a template can specify tabular and graph input format specifications using, e.g., XSD and ShEx, and transformations to and from (liftings and lowerings [5]) the ontology output format using, e.g., XSLT and SPARQL. This means that data can be captured in bulk using XML- or XSD-aware client tools, and efficiently processed using XSLT and/or SPARQL processors, all of which are driven by specifications generated from a template. The process is illustrated and explained in Fig. 2.

Using this framework the ontology engineering task can be split in two more or less distinct responsibilities: one managed by the ontology engineer and the other by the domain matter expert. The main task of the ontology engineer is to design and maintain a library of interconnected templates capable of capturing the knowledge of the domain matter expert at the correct abstraction level and using a vocabulary and format recognisable by the expert. The specificity needed for the ontology engineering task at hand is achieved by iteratively combining and composing basic and more complex templates to result in *user-facing* templates. From such templates, tabular input format specification and transformations may be generated from the template specification, presenting a simple tabular format for the user to fill in which can be transformed directly to OWL ontology format using readily available desktop tools.

This process ensures uniformity and completeness of the captured domain knowledge: completeness, as the template specifies all the attributes that are necessary and variable; and uniformity as the template instances are guaranteed to expand to the desired patterns. The correctness of templates may be secured by consistency checking the prototype ontology resulting from expanding the template, as well as for each of the comprising templates in isolation. Additional syntactic constraints on the input data may be specified for the input formats which also can be used to check completeness of the input data, provided the format specification works under closed-world semantics.

2.3 Implementation

A prototype implementation that interprets the OTTR vocabulary and provides parts of the technical framework presented in Sec. 2.2 is available as an online web application and as a feature-limited standalone Java application from `http://www.ottr.xyz`.

Provided an OTTR template, specified with the IRI query parameter `?tpl`, the web implementation serves a variety of specifications and instances over HTTP: the template specification, the expansion, lifting and lowering queries as different types of SPARQL queries, and a simple XSD format and XML sample. Some services allow template instances to be created by providing argument values as IRI query parameters. With these services, the template and its different format specifications may be directly identified and used by other specifications, e.g., in OWL ontology `owl:import` statements.

Example 7. We encourage the reader to visit `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/i17/partlength` for a display of the PartLength template of Ex. 4. The service lists the template's parameters with type information and containing template instances, together with links to all the other available services of the web application.

3 Ontology Design Patterns vs. Ontology Templates

As argued in the introduction, we believe that ODPs in their current form are not practical ontology building blocks, in the sense of being actively and directly applicable in the engineering of OWL ontologies. Rather they are conceptual building blocks that (only[8]) represent best practices of common modelling challenges—which of course is extremely useful. The practical ways of using an ODP in OWL ontology development are however limited: the natural possibilities are either to import its OWL implementation, which includes the whole pattern as-is, or by cloning (parts of) it. The included pattern may be further engineered with different techniques such as specialisation and generalisation [20]. The XDP [7] tool offers the possibility to instantiate ODPs using so called template-based and specialisation-based techniques [8]. However, in all these cases the process is largely manual and does not scale.

Reasonable Ontology Templates offer a technical framework that allows patterns, as represented by ODPs, to be applied to OWL ontology engineering in an efficient and transparent manner, avoiding unnecessary manual intervention. Comparing ODPs and OTTRs to software engineering, we believe that ODPs play the role of software engineering patterns, "representing general reusable solution to a commonly occurring problem within a given context[, but] not a finished design that can be transformed directly into source or machine code" [22]. In contrast, we think of a set of OTTRs as representing an application programming interface (API) for OWL ontology engineering, like the OpenGL API[9] does for graphics rendering,

[8] In the literature ODPs are often represented as both best practice modelling patterns *and* practical building blocks.

[9] URL: `https://www.opengl.org/`

providing precise and transparent abstractions over the underlying OWL syntax that are directly applicable in the construction of OWL ontologies.

The idea of an API for OWL patterns has been implemented by representing the structural specification of OWL 2 to RDF graphs [19] as a set of OTTR templates. The purpose is to demonstrate that OWL ontologies may be represented completely by a set of OTTR instances, which, in its terse list input format, are arguably more readable than the RDF serialisation of OWL axioms. The EquivObjectUnionOf of Ex. 5 is an example of an OTTR template of an OWL axiom. Other examples are found in Fig. 3a. These templates, including other templates of different maturity, are published in an online library of OTTRs available at `http://library.ottr.xyz` and backed by open git repositories at `http://www.gitlab.com/ottr`.

3.1 Building Ontologies and Linked Datasets with OTTRs

To illustrate how OTTRs can be applied for ontology modelling and publication of linked data, we now demonstrate how OTTRs can be used to construct the Chess Game ODP [15] and to produce linked data representations of chess games, cf. [14].

The Chess Game ODP [15] is presented as a worked example for modelling with ODPs, using them to construct a chess game ontology intended to be used for describing chess games, i.e., who the players were, the end result, the list of moves, the chess opening, and where the game took place. To this end, the authors use adapted versions of the Agent Role and Event ODP, where the Event ODP extends the Agent Role ODP. They also make use of implicit OWL axiom macros for expressing *scoped domains* and *ranges*, and regular axioms like cardinality restrictions. Although the exposition and the graphical depictions of the chess game pattern are clear, its axiomatisation, and hence its OWL implementation,[10] reveals shortcomings of the presentation. These problems stem mainly from the fact that no abstraction mechanism that can encapsulate patterns and present clear interfaces for use is available: The agent role and event patterns are not clearly identified and encapsulated, and it is not clear which axioms belong to which patterns. (This is only made clear by the document formatting.) These patterns are also specialised, but it is difficult to identify exactly how, e.g., for which parts of the pattern are

[10] See `http://ontologydesignpatterns.org/wiki/Submissions:ChessGame`

specialisations introduced. The scoped domain and range axioms are often used, and usually in pairs, but this is not easy to spot.

A sample of the OTTR templates used to reconstruct the Chess Game pattern is found in Fig. 3. The ScopedDomainRange template in Fig. 3b illustrates the how to basic OWL axiom templates (found in Fig. 3a) can be combined. The Event_{10} template succinctly presents that its definition relies on the $\mathsf{AgentRole}_5$ template and other templates, some which represent regular OWL axioms like existential restrictions ($\mathcal{T}_{\sqsubseteq,\exists}(\cdot,\cdot,\cdot)$). Note that all of these OTTR templates represent template-based instantiatations [8], which can be seen by the fact that all vocabulary elements are parameterised; the user can (and must) introduce the vocabulary, the only fixed vocabulary is the logical vocabulary. Fig. 3c contains two specialisation-based [8] OTTR templates, where (some) fixed non-logical vocabulary elements are specialised by template arguments. In our modelling of the chess game pattern we use template-based OTTRs to introduce the basic vocabulary of the pattern. This vocabulary is then used in specialisation-based OTTRs to provide the user with patterns which do not necessarily represent any "self-contained semantic unit", but merely represents a combination of often used axioms packaged in a template to avoid tedious repetitions. From this observation it follows that template-based OTTRs arguably better fit the idea of a publicly available API, while specialisation-based OTTR templates are naturally closer tied to specific ontology models. The complete Chess Game OTTR can be found at `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/ChessGame.ttl`.

Krisnadhi et al. [14] demonstrate how linked data representations can be supported by ODPs with the central operations of *pattern flattening* and *view expansion* (in Sec. 2.2 we call these operations *lowering* and *lifting*) implemented using SPARQL UPDATE queries to transform compact linked data formats (views) to pattern representations, and vice versa. They also show how graph pattern conformance can be checked using SHACL [13].

This resembles the framework described in Sec. 2.2. However, a benefit of our approach is that, while all queries and format descriptions in the referenced work appears to be hand-crafted, similar lifting and lowering format descriptions and transformation can be automatically generated given an appropriate template specification. To demonstrate the abilities of OTTR templates for linked data publication, we have built the user-facing iChessGameReport template with which individual chess games may be expressed. (This template, and its nested templates, are equal in form to the Chess Game pattern templates, but are designed to take individuals and data values as input, rather then classes and properties.) The template may be found at `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/iChessGameReport`. From this page different lifting and lowering queries and formats are available, as described in Sec. 2.3. Using the template instance format we can now compactly represent chess game instances which may be expanded to a full OWL pattern. Fig. 3d contains an instance of the iChessGameReport template, available at `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/iChessGameReportExample`. Note that template instances can be regarded as "standardised" pattern views, representing patterns in a compact format using a specific vocabulary. We do recognise that this format may not be fit for linked data publication and that user-defined template views are necessary; see also the future work section in Sec. 5.

Finally, we note that OTTR templates can also be used to identify pattern instantiations or template instances. For instance, using the generated SPARQL query from the ScopedDomainRange we can successfully extract all the macro applications from the published version of the Chess Game ODP [14].

4 Discussion and Related Work

We now highlight the main benefits and shortcomings that are inherent to the template mechanism and the representation language of OTTRs, and compare them with related work. As for benefits:

- OTTRs provide a simple, but powerful abstraction mechanism based on the well-known concept of nested non-cyclic macros and syntactic substitutions. This allows complex ontology expressions to be compactly represented by a naturally compositional structure which we believe supports more efficient construction and maintenance of ontologies following "don't repeat yourself" (DRY) principles.
- OTTR templates lets patterns to be explicitly identified as such and clearly encapsulated. This improves provenience and interoperability between ontologies using the same or related templates, as patterns need not be discovered.
- The implicit mapping between the template head and the body provides the basics for an extensible framework for handling semantic data that

can represent and transform data on and between different formats and abstractions.

- Templates are formally defined as parameterised ontologies. This allows the semantics of the pattern to be verified using regular ontology reasoners. Furthermore, it makes the organisation of templates and the study of relations between them essentially an extension of the same well-studied issues regarding ontologies, and familiar terminology and theoretical machinery can be reused.
- OTTRs can be compactly represented in RDF as OWL ontologies using the OTTR vocabulary. This allows us to leverage the stack of existing W3C languages and tools, such as ontology editors and reasoners.
- As the expansion mechanism is based on syntactic substitutions, templates can take any RDF resource as input, i.e., classes, properties, individuals and data values, including even the resources from the "logical" OWL and RDF(S) vocabularies.

As for shortcomings, some of the benefits have a negative dual side:

- The simple nature of syntactic macros leaves our templates mechanism with limited expressivity: for instance, they do not contain loop structures or conditionals and they do not return values. For a specific example, we cannot currently apply a template to all the subclasses of a given class. Current and future work is directed at allowing for more complex expressions, and at precisely delimiting the expressive power of OTTRs.
- The compact representation of OTTR templates as RDF graphs using the notion of graph neighbourhood, results in a somewhat implicitly defined head and body of the template. This again requires that an RDF document can only contain a single template. It would be convenient to be able to collect multiple templates in one document, and to package templates which only are used by one ontology together with that ontology.
- In the RDF representation of OTTRs regular RDF resources play the role of variables. This means that special care must be taken when minting parameter variables, since upon instantiating the template *all* occurrences of the parameter variable will be replaced with the argument value. Elements from established vocabularies, such as the OWL vocabulary [19], should be avoided as variables or used with extreme care. Less obvious potential problem variable values are

literals, where the same value may unintentionally be used in different contexts, e.g., as cardinality restrictions on properties.

A predecessor and inspiration to the current form of templates dates back to 2008 [12]. A recent paper by authors of the current paper presents a formal definition of templates and investigates their formal properties: using templates as macros, queries, and for data exchange; and reasoning over templates [6]. The paper at hand is the first account of the practical aspects of OTTR templates.

The Ontology Pre-Processor Language (OPPL) [11] is similar in function, but different in form to OTTRs. Like OTTRs, OPPL patterns are parameterised ontology expressions which can be nested and can specify pattern instances and patterns directly in OWL ontologies. OPPL is a more powerful language than the template mechanism allowing OPPL patterns to return values, which supports a more elegant composition of patterns. On the other side, as OPPL was originally designed as an ontology manipulation language for adding and removing ontology axioms, OPPL patterns are expressed as a series of OWL axiom insertions. This, and the fact that the OPPL pattern is represented "in verbatim" as an OPPL script in OWL annotation properties, places the pattern out of reach for ontology reasoners and requires the correctness of the pattern to be checked by reasoning on the effects of applying the pattern, rather that the pattern itself. Also, it is not clear if formal semantics for OPPL patterns are developed. The application focus of OPPL is somewhat different from OTTRs: OPPL patterns are intended to be fully expanded once they are used in the ontology. In contrast, we believe that OTTR template instances can appear in ontologies as instances lifted or lowered to the abstraction level suited for the given user. For instance, an ontology expert may prefer to examine an ontology formatted as a set of OWL axiom OTTR templates, while the domain experts might prefer to see only the user-facing template instances. Additionally, OPPL patterns are limited to OWL expressions in Manchester syntax [10], while OTTRs supports RDF macros and is designed to be applicable in a larger framework, cf. Sec. 2.2.

XDP [7], built on top of WebProtégé, provides a convenient graphical tool for selecting and instantiating templates using a template-based or specialisation-based approach, but does not offer additional capabilities for ODP instantiation at scale.

The M^2 mapping language [18] extends the OWL Manchester syntax [10] with ontology pattern descriptions to include direct references into spreadsheets for translating spreadsheet data into ontologies. It has hence a more narrow focus than OTTRs, but makes direct use of the underlying representation language in a similar manner as OTTRs.

Taking a broader approach, Tawny-OWL [16] provides an environment for building OWL ontologies using Clojure, with all the advantages of using a fully fledged programming language.

5 Conclusion and Future Work

We have presented the simple, but novel technique of using RDF(S) and OWL for representing *Reasonable Ontologies Templates (OTTRs)*. OTTRs are ontology macros or templates that provide an extensible and transparent framework for creating and using ontology design patterns in the design and construction of ontologies. Templates are in essence n-ary relations that relate a simple tabular input format, defined by its template head, to a rich ontological structure in the template body, possibly via compositions of other templates. From the template head, different tabular data input formats may be generated. This allows ontology experts to design "user-facing" templates for the purpose of collecting domain knowledge from expert users on tabular format. The ontological definition of the template is produced by expanding the template to a regular ontology. Bulk transformation specifications of the input data to ontology format may also be generated from the template. Templates are formulated in OWL using a special purpose OTTR OWL vocabulary. By virtue of being OWL ontologies, templates may be shared, reused, and debugged using existing semantic web technologies and tools. Additionally, the OTTR vocabulary supports simple type checking and terse formats for template specifications. A prototype implementation for expanding and generating various specifications from templates is available online at http://www.ottr.xyz.

Future work. The present proposal for templates has been developed in close interaction with industrial user communities, and we intend to apply it to various existing enterprise ontologies in the immediate future. This will serve to evaluate, verify and refine the concept, and will help us develop an efficient and reliable set of tools and web services. We believe that templates can be important for development and use of open, validated modelling patterns, as is required for shared models, and for enabling ontology-based collaboration. In order to develop templates that cover typical needs of industrial users, we will work with standardisation bodies and make these templates available through a public repository. This should lower the cost of translating existing data into ontology, opening up the benefits of ontology-based methods to new users.

To support this work, tools and methods for constructing, structuring and managing templates are necessary. To this end, we intend to further develop the prototype implementation to support more input representation and validation formats, such as spreadsheets and RDF graph shape validations, and to develop a Protégé plugin for developing and applying OTTR templates to ontology development.

We also intend to continue the initial efforts on the logical properties of templates as is found in [6]. A template can dually be regarded as a macro or as a (higher-order) query; whether one is asking for a pattern to be added to an ontology or asking for occurrences of the pattern in the ontology is a difference of use of the template, and not of the template itself. Furthermore, this duality makes it easy to extend the use of templates to adding a pattern conditionally on another pattern occurring in the ontology, or to use templates as constraints on ontologies. The latter observation can for instance be used to implement pattern views, allowing template instances to be specified using a custom vocabulary. A different problem is if this possible to (elegantly) represent in OWL.

References

1. E. Blomqvist, K. Hammar, and V. Presutti. *Engineering Ontologies with Patterns – The eXtreme Design Methodology*, chapter 2, pages 23–50. Volume 025 of Hitzler et al. [9], 2016.
2. D. Brickley and R. Guha. RDF Schema 1.1. Technical report, W3C, 2014.
3. R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Technical report, W3C, 2014.
4. N. Drummond et al. Putting OWL in Order: Patterns for Sequences in OWL. In *OWLED*, 2006.
5. J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. Technical report, W3C, 2007.
6. H. Forssell, D. P. Lupp, M. G. Skjæveland, and E. Thorstensen. Reasonable Macros for Ontology Construction and Maintenance. In *DL Workshop*, 2017.
7. K. Hammar. Ontology Design Patterns in WebProtege. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, 2015.
8. K. Hammar and V. Presutti. Template-Based Content ODP Instantiation. Workshop on Ontology and Semantic Web Patterns, WOP 2016.
9. P. Hitzler et al., editors. *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, volume 025. IOS Press, Amsterdam, 2016.
10. M. Horridge and P. F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax. Technical report, W3C, 2012.

11. L. Iannone, A. L. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009.
12. J. W. Klüwer, M. G. Skjæveland, and M. Valen-Sendstad. ISO 15926 templates and the Semantic Web. W3C Workshop on Semantic Web in Oil & Gas Industry, 2008.
13. H. Knublauch and D. Kontokostas. Shapes Constraint Language (SHACL). Technical report, W3C, 2017.
14. A. Krisnadhi et al. *Ontology Design Patterns for Linked Data Publishing*, chapter 10, pages 201–232. Volume 025 of Hitzler et al. [9], 2016.
15. A. Krisnadhi and P. Hitzler. *Modeling With Ontology Design Patterns: Chess Games As a Worked Example*, chapter 1, pages 3–21. Volume 025 of Hitzler et al. [9], 2016.
16. P. Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. In *OWLED*, 2013.
17. B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. Technical report, W3C, 2012.
18. M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. M2: A Language for Mapping Spreadsheets to OWL. In *OWLED*, 2010.
19. P. F. Patel-Schneider and B. Motik. OWL 2 Web Ontology Language Mapping to RDF Graphs. Technical report, W3C, 2012.
20. V. Presutti and A. Gangemi. Content Ontology Design Patterns As Practical Building Blocks for Web Ontologies. In *ER*, pages 128–141. Springer, 2008.
21. D. Vrandečić. Explicit knowledge engineering patterns with macros. In *Proceedings of the Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, 2005.
22. Wikipedia. Software design pattern—Wikipedia, the free encyclopedia, 2017. [Online; accessed 27-July-2017].

$$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{A}, \mathsf{R}, \mathsf{B}) :: \{ \mathsf{A} \sqsubseteq \exists \mathsf{R}.\mathsf{B} \} \qquad \mathcal{T}_{\sqsupseteq,\exists}(\mathsf{A}, \mathsf{R}, \mathsf{B}) :: \{ \exists \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$$

$$\mathcal{T}_{\sqsubseteq,\forall}(\mathsf{A}, \mathsf{R}, \mathsf{B}) :: \{ \mathsf{A} \sqsubseteq \forall \mathsf{R}.\mathsf{B} \} \qquad \mathcal{T}_{\sqsupseteq,\forall}(\mathsf{A}, \mathsf{R}, \mathsf{B}) :: \{ \forall \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$$

$$\mathcal{T}_{\sqsubseteq,=}(\mathsf{A}, \mathsf{i}, \mathsf{R}, \mathsf{B}) :: \{ \mathsf{A} \sqsubseteq =_{\mathsf{i}} \mathsf{R}.\mathsf{B} \} \qquad \mathcal{T}_{\sqsupseteq,=}(\mathsf{A}, \mathsf{i}, \mathsf{R}, \mathsf{B}) :: \{ =_{\mathsf{i}} \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$$

$$\mathsf{DisjointClasses}(\langle \mathsf{C}_1, \mathsf{C}_2, \ldots \rangle) :: \{ \mathtt{DisjointClasses}(\mathsf{C}_1, \mathsf{C}_2, \ldots) \}$$

(a) Basic OWL axioms represented as OTTR templates.

$$\begin{aligned}
&\mathsf{ScopedDomainRange}(\mathsf{R}, \mathsf{A}, \mathsf{B}) :: \{ \mathcal{T}_{\sqsupseteq,\exists}(\mathsf{A}, \mathsf{R}, \mathsf{B}), \mathcal{T}_{\sqsubseteq,\forall}(\mathsf{A}, \mathsf{R}, \mathsf{B}) \}\\[4pt]
&\mathsf{AgentRole}_5(\mathsf{AgentRole}, \mathsf{RoleProvider}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy}) :: \{\\
&\quad \mathsf{Range}(\mathsf{providesRole}, \mathsf{AgentRole}),\\
&\quad \mathsf{ScopedDomainRange}(\mathsf{performedBy}, \mathsf{AgentRole}, \mathsf{Agent}),\\
&\quad \mathcal{T}_{\sqsubseteq,\exists}(\mathsf{AgentRole}, \mathsf{providesRole}^{-}, \mathsf{RoleProvider})\\
&\quad \mathcal{T}_{\sqsubseteq,\exists}(\mathsf{AgentRole}, \mathsf{performedBy}, \mathsf{Agent}),\\
&\quad \mathsf{DisjointClasses}(\langle \mathsf{AgentRole}, \mathsf{Agent} \rangle)\}\\[4pt]
&\mathsf{Event}_{10}(\mathsf{Event}, \mathsf{subEventOf}, \mathsf{AgentRole}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy},\\
&\qquad \mathsf{Place}, \mathsf{atPlace}, \mathsf{TemporalExtent}, \mathsf{atTime}) :: \{\\
&\quad \mathsf{AgentRole}_5(\mathsf{AgentRole}, \mathsf{Event}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy})\\
&\quad \mathcal{T}_{\sqsubseteq,\exists}(\mathsf{Event}, \mathsf{atPlace}, \mathsf{Place}),\\
&\quad \mathcal{T}_{\sqsubseteq,\exists}(\mathsf{Event}, \mathsf{atTime}, \mathsf{TempExt}),\\
&\quad \mathsf{ScopedDomainRange}(\mathsf{atPlace}, \mathsf{Event}, \mathsf{Place}),\\
&\quad \mathsf{ScopedDomainRange}(\mathsf{atTime}, \mathsf{Event}, \mathsf{Time}),\\
&\quad \mathsf{ScopedDomainRange}(\mathsf{subEventOf}, \mathsf{Event}, \mathsf{Event}),\\
&\quad \mathsf{DisjointClasses}(\langle \mathsf{Event}, \mathsf{Place}, \mathsf{TempExt}, \mathsf{AgentRole}, \mathsf{Agent} \rangle)\}
\end{aligned}$$

(b) Template-based OTTR templates: ScopedDomainRange, AgentRole and Event.

$$\begin{aligned}
&\mathsf{AgentRole}_2(\mathsf{xAgentRole}, \mathsf{xRoleProvider}) :: \{\\
&\quad \mathsf{xAgentRole} \sqsubseteq \text{AgentRole},\\
&\quad \mathcal{T}_{\sqsubseteq,\exists}(\mathsf{xRoleProvider}, \text{providesAgentRole}, \mathsf{xAgentRole}),\\
&\quad \mathcal{T}_{\sqsubseteq,=}(\mathsf{xAgentRole}, 1, \text{providesAgentRole}^{-}, \mathsf{xRoleProvider})\}\\[4pt]
&\mathsf{Event}_2(\mathsf{xEvent}, \mathsf{xAgentRole}) :: \{\\
&\quad \mathsf{xEvent} \sqsubseteq \text{Event},\\
&\quad \mathsf{AgentRole}_2(\mathsf{xAgentRole}, \mathsf{xEvent})\}
\end{aligned}$$

(c) Specialisation-based OTTR templates for AgentRole and Event patterns.

```
[] ottr:templateRef <http://draft.ottr.xyz/chess/iChessGameReport> ;
   ottr:withValues (  "WCh 2013" "Chennai IND" "2013.11.09" "Carlsen, Magnus"
      "Anand, Viswanathan" "1/2-1/2" "2870" "2775" "A07" ( "Nf3" "d5" [...] ))) .
```

(d) OTTR template instance of the ChessGameReport template, including only two chess moves.

Fig. 3: OTTR templates used for the Chess Game pattern and for linked data publication.

Paper 4

Practical Ontology Pattern Instantiation, Discovery, and Maintanenence with Reasonable Ontology Templates

Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. "Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates." In: *The Semantic Web – ISWC 2018*. Ed. by Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl. Springer International Publishing, 2018, pp. 477–494. ISBN: 978-3-030-00671-6 [P4]

Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates

Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell

{martige,danielup,leifhka,jonf}@ifi.uio.no
Department of Informatics, University of Oslo

Abstract. Reasonable Ontology Templates (OTTR) is a language for representing ontology modelling patterns in the form of parameterised ontologies. Ontology templates are simple and powerful abstractions useful for constructing, interacting with, and maintaining ontologies. With ontology templates, modelling patterns can be uniquely identified and encapsulated, broken down into convenient and manageable pieces, instantiated, and used as queries. Formal relations defined over templates support sophisticated maintenance tasks for sets of templates, such as revealing redundancies and suggesting new templates for representing implicit patterns. Ontology templates are designed for practical use; an OWL vocabulary, convenient serialisation formats for the semantic web and for terse specification of template definitions and bulk instances are available, including an open source implementation for using templates. Our approach is successfully tested on a real-world large-scale ontology in the engineering domain.

1 Introduction

Constructing sustainable large-scale ontologies of high quality is hard. Part of the problem is the lack of established tool-supported best-practices for ontology construction and maintenance. From a high-level perspective [12], an ontology is built through three iterative phases:

1. Understanding the target domain, e.g., the domain of pizzas
2. Identifying relevant abstractions over the domain, e.g., "Margherita is a particular Italian pizza with only mozzarella and tomato"
3. Formulating the abstractions in a formal language like description logics; here an adapted excerpt taken from the well-known Pizza

ontology tutorial:[1]

$$\mathsf{Margherita} \sqsubseteq \mathsf{NamedPizza} \sqcap \exists\, \mathsf{hasCountryOfOrigin}.\{\mathsf{Italy}\} \tag{1}$$

$$\mathsf{Margherita} \sqsubseteq \exists\, \mathsf{hasTopping}.\mathsf{Mozzerella} \sqcap \exists\, \mathsf{hasTopping}.\mathsf{Tomato} \tag{2}$$

$$\mathsf{Margherita} \sqsubseteq \forall\, \mathsf{hasTopping}.(\mathsf{Mozzerella} \sqcup \mathsf{Tomato}) \tag{3}$$

This paper concerns the third task and targets particularly the large gap that exists between how domain knowledge facts are naturally expressed, e.g., in natural language, and how the same information must be recorded in OWL. The cause of the gap is the fact that OWL at its core supports only unary and binary predicates (classes and properties), and offers no real mechanism for user-defined abstractions with which recurring modelling patterns can be captured, encapsulated, and instantiated. The effect is that every single modelled statement no longer remains a coherent unit but must be broken down into the small building blocks of OWL. And as there is no trace from the original domain statement to the ontology axioms, the resulting ontology is hard to comprehend and difficult and error-prone to manage and maintain.

As a case in point, the Pizza ontology contains 22 different types of pizzas, all of which follow the same pattern of axioms as the encoding of the Margherita pizza seen above. For both the user of the ontology and the ontology engineer this information is opaque. The axioms that make out the instances of the pattern are all kept in a single set of OWL axioms or RDF triples in the same ontology document. Since the pizza pattern is not represented as a pattern anywhere, tasks that are important for the efficient use and management of the ontology, such as finding pattern instances and verifying consistent use of the pattern, i.e., understanding the ontology and updating the pattern, may require considerable repetitive and laborious effort.

In this paper we present *Reasonable Ontology Templates* (OTTR), a language for representing ontology modelling patterns as parameterised ontologies, implemented using a recursive non-cyclic macro mechanism for RDF. A pattern is instantiated using the macro's succinct interface. Instances may be *expanded* by recursively replacing instances with the pattern they represent, resulting in an ordinary RDF graph. Section 2 presents the fundamentals of the OTTR language and exemplifies its use on the pizza pattern. Ontology templates are designed to be practical and

[1] https://protege.stanford.edu/ontologies/pizza/pizza.owl

versatile for constructing, using and maintaining ontologies; the practical aspects of using templates are covered in Section 3. Section 4 concerns the maintenance of ontology template libraries. It presents methods and tools that exploit the underlying theoretical framework to give sophisticated techniques for maintaining template libraries and ultimately the ontologies built from those templates. We define different relations over templates and show how these can be used to define and identify imperfections in a template library, such as redundancy, and to suggest improvements of the library. We believe ontology templates can be an important instrument for improving the efficiency and quality of ontology construction and maintenance. OTTR templates allow the design of the ontology, represented by a relatively small library of templates, to be clearly separated from the bulk content of the ontology, specified by a large set of template instances. This, we believe, supports better delegation of responsibility in ontology engineering projects, allowing ontology experts to build and manage a library of templates and domain experts to provide content in the form of structurally simple template instances. To support this claim we report in Section 5 from successful experiments on the use of ontology templates to build and analyse Aibel's large-scale Material Master Data (MMD) ontology. We compare our work with existing approaches in Section 6 and present ideas for future work in Section 7.

2 Reasonable Ontology Templates Fundamentals

In this section we develop the fundamentals for OTTR templates as a generic macro mechanism adapted for RDF.

An *OTTR template* T consists of a *head*, head(T), and a *body*, body(T). The body represents a *parameterised* ontology pattern, and the head specifies the template's name and its parameters, param(T). A *template instance* consists of a template name and a list of arguments that matches the template's specified parameters and represents a replica of the template's body pattern where parameters are replaced by the instance's arguments. The template body comprises only template instances, i.e., the template pattern is recursively built up from other templates, under the constraint that cyclic template dependencies are not allowed. There is one special *base template*, the TRIPLE *template*, which takes three arguments. This template has no body but represents a single RDF triple in the obvious way. *Expanding* an instance is the process of recursively replacing instances with the pattern they represent. This process terminates with an expression containing only Triple template instances, hence representing an RDF graph.

Example 1. The SubClassOf template is a simple representation of the rdfs:subClassOf relationship. It has two parameters, ?sub and ?super, and a body containing a single instance of the Triple template.

$$\overbrace{\underbrace{\textsc{SubClassOf}}_{\text{name}}(\underbrace{\text{?sub, ?super}}_{\text{parameters}})}^{\text{head}} \;::\; \overbrace{\underbrace{\textsc{Triple}(\text{?sub, rdfs:subClassOf, ?super})}_{\text{instance}}}^{\text{body}} \;.$$

An example instance of this template is SubClassOf(:Margherita, :NamedPizza); it expands, in one step, to a single Triple instance which represents the (single triple) RDF graph ⟨:Margherita, rdfs:subClassOf, :NamedPizza⟩.

Each template parameter has a *type* and a *cardinality*. (If these are not specified, as in Ex. 1, default values apply.) The type of the parameter specifies the permissible type of its arguments. The available types are limited to a specified set of classes and datatypes defined in the XSD, RDF, RDFS, and OWL specifications, e.g,. xsd:integer, rdf:Property, rdfs:Resource and owl:ObjectProperty. The OWL ontology at `ns.ottr.xyz/templates-term-types.owl` declares all permissible types and organises them in a hierarchy of *subtypes* and *incompatible* types, e.g., owl:ObjectProperty is a subtype of rdf:Property, and xsd:integer and rdf:Property are incompatible. The most general and default type is rdfs:Resource. This information is used to type check template instantiations; a parameter may not be instantiated by an argument with an incompatible type.

The cardinality of a parameter specifies the number of required arguments to the parameter. There are four cardinalities: *mandatory* (written 1), *optional* (?), *multiple* (+), and *optional multiple* (∗), which is shorthand for ? and + combined. *Mandatory* is the default cardinality. Mandatory parameters require an argument. Optional parameters permit a missing value; *none* designates this value. If *none* is an argument to a mandatory parameter of an instance, the instance is ignored and will not be included in the expansion. A parameter with cardinality *multiple* requires a list as its argument. Instances of templates that accept list arguments may be used together with an *expansion mode*. The mode indicates that the list

$$\text{ObjectAllValuesFrom}(?X : 1\ \mathsf{nonLiteral}, ?P : 1\ \mathsf{property}, ?R : 1\ \mathsf{nonLiteral})$$
$$::\ (?X, \text{rdf:type}, \text{owl:Restriction}), (?X, \text{owl:onProperty}, ?P), (?X, \text{owl:allValuesFrom}, ?R)\ .$$

$$\text{SubObjectAllValuesFrom}(?X : 1\ \mathsf{class}, ?P : 1\ \mathsf{objectProperty}, ?R : 1\ \mathsf{class})$$
$$::\ \text{SubClassOf}(?X, \_\text{:b1}), \text{ObjectAllValuesFrom}(\_\text{:b1}, ?P, ?R)\ . \tag{4}$$

$$\text{ObjectUnionOf}(?X : 1\ \mathsf{nonLiteral}, ?union : +\ \mathsf{class})$$
$$::\ (?X, \text{rdf:type}, \text{owl:Class}), (?X, \text{owl:unionOf}, ?union)\ .$$

Fig. 1: Basic OWL OTTR templates

arguments will in the expansion be used to generate multiple instances of the template. There are two modes: *cross* (written x) and *zip* (z). The instances to be generated are calculated by temporarily considering all arguments to the instance as lists, where single value arguments become singular lists. In cross mode, one instance per element in the cross product of the temporary lists is generated, while in zip mode, one instance per element in the zip of the lists is generated. List arguments used without an expansion mode behave just like regular arguments. Parameters with cardinality *optional multiple* also accept *none* as a value.

Example 2. Fig. 1 contains three examples of OTTR templates that capture basic OWL axioms or restrictions, and exemplify the use of types and cardinalities. The template SubObjectAllValuesFrom represents the pattern $?X \sqsubseteq \forall ?P.?R$ and is defined using the SubClassOf and ObjectAllValuesFrom templates. Note that we allow a Triple instance to be written without its template name. The parameters of SubObjectAllValuesFrom are all mandatory, and have respectively the types class, objectProperty and class. The ObjectUnionOf template represents a union of classes. Here the parameter types are nonLiteral and class, where the latter has cardinality *multiple* in order to accept a list of classes. The type of the first parameter, nonLiteral, prevents an argument of type literal.

Example 3. The pizza pattern presented in the introduction is represented as an OTTR template in Fig. 2(a) together with two example instances. The template takes three arguments: the pizza, its optional country of origin, and its list of toppings. The cross expansion mode (×) on the SubObjectSomeValuesFrom instance causes it to expand to one instance per topping in the list of toppings, e.g., for the first example instance:

– SubObjectSomeValuesFrom(:Margherita, :hasTopping, :Mozzarella) and
– SubObjectSomeValuesFrom(:Margherita, :hasTopping, :Tomato),

creating an existential value restriction axiom for each topping, which results in the set of axioms seen in (2) of the pizza pattern in Section 1. By joining SubObjectAllValuesFrom and ObjectUnionOf with a blank node (_:b1), we get the universal restriction to the union of toppings (3). Note that the list of toppings is used both to create a set of existential axioms *and* to create a union class. The optional ?Country parameter behaves so that the SubObjectHasValue instance is not expanded but removed in the case that ?Country is *none*. The first NamedPizza instance in the figure represents exactly the same set of axioms as the listing in Section 1.

We conclude this section with the remark that it is in principle possible to choose a "base" other than RDF for OTTR templates, with suitable changes to typing and to which templates are designated as base templates. For instance, we could let templates such as SubClassOf, SubObjectAllValuesFrom, etc. be our base templates, to form a foundation based on OWL. These templates could then be directly translated into corresponding OWL axioms in some serialisation format. (An OTTR template can also be defined as a parameterised Description Logic knowledge base [2].) We have chosen here to base OTTR templates on RDF as this makes a simpler base, and broadens the application areas of OTTR templates, while still supporting OWL.

3 Using Ontology Templates

In this section we present the resources available to enable efficient and practical use of ontology templates: serialisation formats for templates and template instances, tools, formats and specifications that can be generated from templates, and online resources.

Languages. There are currently three serialisation formats for representing templates and template instances: stOTTR, wOTTR, and tabOTTR.

stOTTR[2] is the format used in the examples of Section 2 and is developed to offer a compact way of representing templates and instances that is also easy to read and write.

[2] URL: https://gitlab.com/ottr/language/stOTTR/

```
NamedPizza(
    ?Name : 1 class,
    ?Country : ? individual,
    ?Toppings : + class)
  ::
    SubClassOf(?Name, :NamedPizza),
    SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country),
    SubObjectAllValuesFrom(?Name, :hasTopping, _:b1),
    ObjectUnionOf(_:b1, ?Toppings),
    × | SubObjectSomeValuesFrom(?Name, :hasTopping, ?Toppings) .

NamedPizza(:Margherita, :Italy, ⟨:Tomato, :Mozzerella⟩)
NamedPizza(:Grandiosa, none, ⟨:Tomato, :Jarlsberg, :Ham, :SweetPepper⟩)
```

(a) stOTTR serialisation and instances

```
<http://draft.ottr.xyz/pizza/NamedPizza> a ottr:Template ;
 ottr:hasParameter
  [ ottr:index 1 ; ottr:classVariable :pizza ] ,
  [ ottr:index 2 ; ottr:individualVariable :country;
      ottr:optional true ] ,
  [ ottr:index 3 ; ottr:listVariable (:toppings) ]  .
    ### body:
[] ottr:templateRef t-owl-axiom:SubClassOf ;
     ottr:withValues ( :pizza p:NamedPizza ) .
[] ottr:templateRef t-owl-axiom:SubObjectHasValue ;
     ottr:withValues ( :pizza p:hasCountryOfOrigin :country ) .
[] ottr:templateRef t-owl-axiom:SubObjectAllValuesFrom ;
     ottr:withValues ( :pizza p:hasTopping _:alltoppings ) .
[] ottr:templateRef t-owl-rstr:ObjectUnionOf ;
     ottr:withValues ( _:alltoppings (:toppings) ) .
[] ottr:templateRef t-owl-axiom:SubObjectSomeValuesFrom ;
     ottr:hasArgument [ ottr:index 1; ottr:value :pizza ] ,
        [ ottr:index 2; ottr:value p:hasTopping ] ,
        [ ottr:index 3; ottr:eachValue (:toppings) ] .
```

(b) wOTTR serialisation

```
:pizza  rdfs:subClassOf  p:NamedPizza ,
     [ a                   owl:Restriction ;
       owl:onProperty      p:hasTopping ;
       owl:someValuesFrom   :toppings ] ,
     [ a                  owl:Restriction ;
       owl:onProperty     p:hasTopping
       owl:allValuesFrom    [
           a             owl:Class ;
           owl:unionOf   ( :toppings ) ] ] ,
     [ a                owl:Restriction ;
       owl:onProperty   p:hasCountryOfOrigin ;
       owl:hasValue     :country ] .
p:hasTopping          a   owl:ObjectProperty .
p:hasCountryOfOrigin  a   owl:ObjectProperty .
:toppings             a   owl:Class .
```

(c) Expanded RDF graph

```
SELECT  *
  { ?param1   rdfs:subClassOf     p:NamedPizza ,
      [ owl:onProperty      p:hasTopping ;
        owl:someValuesFrom  param3item ;
        rdf:type            owl:Restriction ] ,
      [ owl:allValuesFrom   [ owl:unionOf  ?param3 ;
                              rdf:type     owl:Class ] ;
        owl:onProperty      p:hasTopping ;
        rdf:type            owl:Restriction ]
    OPTIONAL {
      ?param1   rdfs:subClassOf  [
          owl:hasValue        ?param2 ;
          owl:onProperty      p:hasCountryOfOrigin ;
          rdf:type            owl:Restriction ]
    }
      ?param3   (rdf:rest)*/rdf:first ?param3item
  }
```

(d) SPARQL SELECT query

?param1	?param3item	?param2
p:Margherita	p:MozzarellaTopping	
p:Margherita	p:TomatoTopping	
p:Mushroom	p:MozzarellaTopping	
p:Mushroom	p:MushroomTopping	
p:Mushroom	p:TomatoTopping	
p:Napoletana	p:AnchoviesTopping	p:Italy
p:Napoletana	p:CaperTopping	p:Italy
p:Napoletana	p:MozzarellaTopping	p:Italy
p:Napoletana	p:OliveTopping	p:Italy
p:Napoletana	p:TomatoTopping	p:Italy

(e) Excerpt query results

```
#OTTR  prefix
p  http://www.co-ode.org/ontologies/pizza/pizza.owl#
#OTTR  end
#OTTR  template  http://draft.ottr.xyz/pizza/NamedPizza
pizza          country   toppings
1              2         3
iri            iri       iri+
p:Margherita   p:Italy   p:Tomato|p:Cheese
p:Grandiosa              p:Tomato|p:Jarlsberg|p:Ham|p:Pepper
#OTTR  end
```

(f) tabOTTR instance serialisation

Fig. 2: NamedPizza template and example instances in different serialisations

However, to enable truly practical use of OTTR for OWL ontology engineering, we have developed a special-purpose RDF/OWL vocabulary, called *wOTTR*, with which OTTR templates and instances can be formulated. This has the benefit that we can leverage the existing stack of W3C languages and tools for developing, publishing, and maintaining templates. The wOTTR format supports writing TRIPLE instances as regular RDF triples. This means that a pattern represented by an RDF graph or RDF/OWL ontology can easily be turned into an OTTR template by simply specifying the name of the template and its parameters with the wOTTR vocabulary. Furthermore, this means that we can make use of existing ontology editors and reasoners to construct and verify the soundness of templates. The wOTTR representation has been developed to closely resemble stOTTR. It uses RDF resources to represent parameters and arguments, and RDF lists (which have a convenient formatting in Turtle syntax) for lists of parameters and arguments. The vocabulary is published at `ns.ottr.xyz`. A more thorough presentation of the vocabulary is found in [13].

tabOTTR[3] is developed particularly for representing large sets of template instances in tabular formats such as spreadsheets, and is intended for domain expert use.

Generated queries and format specifications. A template may not only be used as a macro, but also, inversely, as a query that retrieves all instances of the pattern and outputs the result in the tabular format of the template head. From a template we can generate queries from both its expanded and unexpanded body. The expanded version allows us to find instances of a pattern in "vanilla" RDF data, while the unexpanded version can be used to collect and transform (in the opposite direction than of expansion) a set of template instances into an instance of a larger template. The latter form is convenient for validating the proper usage of templates within a library, which we present in Section 4.

We are also experimenting with generating other specifications from a template, for instance XSD descriptions of template heads, and transformations of these formats, e.g., XSLT transformations. The purpose of supporting other formats is to allow for different data input formats and leverage existing tools for input verification and bulk transformation of instance data to expanded RDF, such as XSD validators and XSLT transformation engines.

Tools and Online Resources. Lutra, our Java implementation of the OTTR template macro expander, is available as open source with an LGPL licence at `gitlab.com/ottr`. It can read and write templates and instances of the

[3] URL: `https://gitlab.com/ottr/language/tabOTTR/`

formats described above and expand them into RDF graphs and OWL ontologies, while performing various quality checks such as parameter type checking and checking the resulting output for semantic consistency. Lutra is also deployed as a web application that will parse and display any OTTR template available online. The template may be expanded and converted into all the formats mentioned above, including SPARQL SELECT, CONSTRUCT and UPDATE queries, XSD format, and variants of expansions which include or exclude the head or body.

Also available, at `library.ottr.xyz`, is a "standard" set of ontology templates for expressing common RDF, RDFS, and OWL patterns as well as other example templates. These templates are conveniently presented in an online library that is linked to the online web application.

Example 4. Fig. 2 contains different representations of the NamedPizza template. Fig. 2(b) contains the published version of the template, available at its IRI address: `http://draft.ottr.xyz/pizza/NamedPizza`. Fig. 2(c) contains the expansion of the template body. Fig. 2(d) displays the generated SPARQL query that retrieves instances of the pizza pattern; an excerpt of the results applying the query to the Pizza ontology is given in Fig. 2(e). Fig. 2(f) contains a tabOTTR representation of the two instances seen in Fig. 2(a). We encourage the reader to visit the rendering of the template by the web application at

`osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/pizza/NamedPizza`

and explore the various presentations and formats displayed. An example-driven walk-through of the features of Lutra can be found at `ottr.xyz/event/2018-10-08-iswc/`.

4 Maintenance and Optimisation of OTTR Template Libraries

In this section, we present an initial list and analysis of some of the more central relations between OTTR templates, and discuss their use in template library optimisation. We focus in particular on removing redundancy within a library, where we distinguish two different types of redundancy: a lack of reuse of existing templates, as well as recurring patterns not captured by templates within the library. We present an efficient and automated technique for detecting such redundancies within an OTTR template library.

4.1 OTTR template relations

Optimisation and maintenance of OTTR template libraries is made possible by its solid formal foundation. OTTR syntax makes it possible to formally define relations between OTTR templates which can tangibly benefit the optimisation of a template library. Naturally, there are any number of ways templates can be "related" to one another, and the "optimal" size and shape of a template library is likely to be highly domain and ontology-specific. As such, we do not aspire to a best-practice approach to optimising a template library. Instead, we illustrate the point by defining a few central template relations and demonstrating their usefulness for library optimisation and maintenance, independently of the heuristics used. Here, we limit ourselves to template relations defined syntactically in terms of instances, and do not consider, e.g., those defined in terms of semantic relationships between full expansions of templates. We consider the following template relations:

directly depends (DD) S *directly depends on* T if S's body has an instance of T.
depends (D) *depends* is the transitive closure of *directly depends*.
dependency-overlaps (DO) S *dependency-overlaps* T if there exists a template upon which both S and T directly depend.
overlaps (O) S *overlaps* T if there exist template instances i_S, i_T in body(S) and body(T) and substitutions ρ and η of the parameters of S and T resp. such that $\rho(i_S) = i_T$ and $\eta(i_T) = i_S$.
contains (C) S *contains* T if there exists a substitution ρ of the parameters of T such that $\rho(\text{body}(T)) \subseteq \text{body}(S)$.
equals (E) S *is equal to* T if S contains T and vice versa.

Each of the listed relations is, in a sense, a specialisation of the previous one (except for *DO*, which is a specialisation of *DD* as opposed to *D*). For instance, *DO* imposes no restrictions on the instance arguments, whereas *O* intuitively requires parameters to occur in compatible positions of i_S and i_T.

Example 5. Consider the template library given in Fig. 3(a). All but the BurgerMeal template contain an instance of SubClassOf, hence all pairs of templates except for (AnnotatedPizza, BurgerMeal) have a dependency-overlap. Closer inspection reveals that Burger contains SubObjectAllValuesFrom (4, Fig. 1), due to the instances

```
NamedPizza(?Name : 1 class, ?Country : ? individual, ?Toppings : + class)        (cf. Fig. 2(a)) .

AnnotatedPizza(?Name : 1 class, ?Label : + literal, ?PrefLabel : ? literal, ?Definition : ? literal)
  :: SubClassOf(?Name, :Pizza),
     × | (?Name, rdfs:label, ?Label), (?Name, skos:prefLabel, ?PrefLabel), (?Name, skos:definition, ?Definition) .     (5)

Burger(?Name : 1 class, ?Condiments : + class, ?Label : + literal, ?PrefLabel : ? literal, ?Definition : ? literal)
  :: SubClassOf(?Name, :Burger),
     × | SubObjectSomeValuesFrom(?Name, :hasCondiment, ?Condiments),
     SubClassOf(?Name, _:b2), ObjectAllValuesFrom(_:b2, :hasCondiment, _:b3),                                         (6)
          ObjectUnionOf(_:b3, ?Condiments),                                                                           (7)
     × | (?Name, rdfs:label, ?Label), (?Name, skos:prefLabel, ?PrefLabel), (?Name, skos:definition, ?Definition) .     (8)

BurgerMeal(?Name : 1 class, ?Sides : + class)
  :: SubObjectSomeValuesFrom(?Name, :hasMain, :Burger),
     SubObjectAllValuesFrom(?Name, :hasSide, _:b4), ObjectUnionOf(_:b4, ?Sides) .                                     (9)
```

(a) OTTR template library with redundancies and lack of re-use

```
NamedPizza(?Name : 1 class, ?Country : ? individual, ?Toppings : + class)
  :: SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country),
     NamedFood(?Name, :NamedPizza, ?Toppings, :hasTopping) .                                                          (⋆)

AnnotatedPizza(?Name : 1 class, ?Label : + literal, ?PrefLabel : ? literal, ?Definition : ? literal)
  :: SubClassOf(?Name, :Pizza),
     Annotation(?Name, ?Label, ?PrefLabel, ?Definition) .                                                             (⋆)

Burger(?Name : 1 class, ?Condiments : + class, ?Label : + literal, ?PrefLabel : ? literal, ?Definition : ? literal)
  :: NamedFood(?Name, :Burger, ?Condiments, :hasCondiment),                                                           (⋆)
     Annotation(?Name, ?Label, ?PrefLabel, ?Definition) .                                                             (⋆)

BurgerMeal(?Name : 1 class, ?Sides : + class)
  :: SubObjectSomeValuesFrom(?Name, :hasMain, :Burger),
     SubObjectAllValuesFromUnion(?Name, :hasSide, ?Sides) .                                                           (⋆)
```

New OTTR templates representing previously uncaptured patterns:

```
NamedFood(?Name : 1 class, ?Category : 1 class, ?Extras : + class, ?hasExtra : 1 objectProperty)                      (10)
  :: SubClassOf(?Name, ?Category),
     × | SubObjectSomeValuesFrom(?Name, ?hasExtra, ?Extras),
     SubObjectAllValuesFromUnion(?Name, ?hasExtra, ?Extras)

Annotation(?Name : 1 class, ?Label : + literal, ?PrefLabel : ? literal, ?Definition : ? literal)                      (11)
  :: × | (?Name, rdfs:label, ?Label), (?Name, skos:prefLabel, ?PrefLabel), (?Name, skos:definition, ?Definition) .

SubObjectAllValuesFromUnion(?x : 1 class, ?Property : 1 objectProperty.?RangeList : + class)                          (12)
  :: SubObjectAllValuesFrom(?x, ?Property, _:b1), ObjectUnionOf(_:b1, ?RangeList) .
```

(b) A refactored version of the templates in Fig. 3(a). The refactored templates from Fig. 3(a) are listed first, where a star (⋆) indicates a dependency to a new template, found at the bottom.

Fig. 3: OTTR template library before and after redundancy removal

– SubClassOf(?Name, _:b2)
– ObjectAllValuesFrom(_:b2, :hasCondiment, _:b3)

in Burger (6). (Numbers refer to numbered lines in the figures.) Finally, AnnotatedPizza and Burger overlap, since they both directly depend on the same Triple templates (5)(8). These relationships are depicted in the graph below (dependency relationships omitted for the sake of legibility). Directed/undirected edges depict nonsymmetric/symmetric relations, respectively.

$$\begin{tikzcd}
\text{NamedPizza} \arrow[r, dash, "\text{DO}"] \arrow[d, dash, "\text{DO}"'] \arrow[dr, dash, "\text{DO}"] & \text{BurgerMeal} \arrow[d, dash, "\text{DO}"] & \\
\text{AnnotatedPizza} \arrow[r, dash, "\text{O}"'] & \text{Burger} \arrow[r, "\text{C}"'] & \text{SubObjectAllValuesFrom}
\end{tikzcd}$$

We wish to discuss these relations in the context of *redundancy removal* within an OTTR template library. More specifically, we discuss two types of redundancy:

Lack of reuse is a redundancy where a template S has a contains relationship to another template T, instead of a dependency relationship to T. That is, S duplicates the pattern represented by T, rather than instantiating T. This can be removed by replacing the offending portion of body(S) with a suitable instance of T. A first approach to determining such a lack of reuse makes use of the fact that templates can be used as queries: template S contains T iff T as a query over S yields answers.

Uncaptured pattern is a redundancy where a pattern of template instances is used by multiple templates, but this pattern is not represented by a template. In order to find uncaptured patterns one must analyse in what manner multiple templates depend on the same set of templates. If multiple templates *overlap* as defined above, this is a good candidate for an uncaptured pattern. However, an overlap does not necessarily need to occur for an uncaptured pattern to be present: as demonstrated in the following example, a dependency-overlap can describe an uncaptured pattern that is relevant for the template library.

Example 6. Continuing with our previous example of the library in Fig. 3(a), we find that it contains both an instance of lack of reuse and multiple instances of uncaptured patterns. The containment of SubObjectAllValuesFrom in Burger indicates a lack of reuse, and the overlap of

BURGER and ANNOTATEDPIZZA is an uncaptured pattern which we refactor into the template ANNOTATION (11). By repairing the lack of reuse in BURGER (6) with an instance of SUBOBJECTALLVALUESFROM, there are two dependency-overlaps that represent uncaptured patterns: the instances (6,7)(9), which are refactored into a new template SUBOBJECTALLVALUES-FROMUNION (12), and the dependency-overlap between BURGER and NAMEDPIZZA, which is described by the NAMEDFOOD template (10). These new templates as well as the updated template definitions for the pre-existing ones are given in Fig. 3(b).

4.2 Efficient redundancy detection

Naive methods for improving a template library using the relations as described in the previous section quickly become infeasible for large knowledge bases, as they require expensive testing of unification of all template bodies. We have developed an efficient method for finding lack of reuse and uncaptured patterns, which over-approximates the results of unification. The method uses the notion of a dependency pair, which intuitively captures repeated use of templates without considering parameters: a *dependency pair* $\langle I, T\rangle$ is a pair of a multiset of templates I and a set of templates T, such that T is the set of all templates that directly depend on all templates in I, and have at least as many directly depends relationships to each template in I as they occur in I. The idea is that I represents a pattern used by all the templates in T. In order to also detect patterns containing different TRIPLE instances, we will in this section treat a TRIPLE instance (s, p, o) as a template instance of the form $p(s, o)$ and thus treat p as a template. Note that for a set of dependency pairs generated from a template library, the first element in the pair, i.e., the I, is unique for the set, while the T is generally not unique.

Example 7. Three examples of dependency pairs from the library in Fig. 3(a) are

1. $\langle\{$SUBCLASSOF, SUBCLASSOF, rdfs:label$\}$, $\{$BURGER$\}\rangle$
2. $\langle\{$SUBCLASSOF, OBJECTALLVALUESFROM$\}$,
   $\{$SUBOBJECTALLVALUESFROM, BURGER$\}\rangle$
3. $\langle\{$skos:definition, rdfs:label, skos:prefLabel$\}$, $\{$ANNOTATEDPIZZA, BURGER$\}\rangle$

The first pair indicates that Burger is the only template that directly depends on two occurrences of SubClassOf and one occurrence of rdfs:label. Note that Burger directly depends on other templates too, and these will give rise to other dependency pairs. However there is no other template than Burger that directly depends on this multiset of templates. The second example shows that SubObjectAllValuesFrom and Burger directly depend on the templates SubClassOf and ObjectAllValuesFrom.

One can compute all dependency pairs by starting with the set of dependency pairs of the form $\langle \{i : n\}, T \rangle$ where all templates in T have at least n instances of i, and then compute all possible *merges*, where a merge between two clusters $\langle I_1, T_1 \rangle$ and $\langle I_2, T_2 \rangle$ is $\langle I_1 \cup I_2, T_1 \cap T_2 \rangle$. We have implemented this algorithm with optimisations that ensure we compute each dependency pair only once.

The set of dependency pairs for a library contains all potential lack of reuse and uncaptured patterns in a library. However, note that in the dependency pairs where either I or T has only one element, the dependency pair does not represent a commonly used pattern: If I has only one element then it does not represent a redundant pattern. If T has only one element then the pattern occurs only once. If on the other hand both sets contain two or more elements then the dependency pair might represent a useful pattern to be represented as a template, and we call these *candidate pairs*.

For a candidate pair, there are three cases to consider: 1. the set of instances does not form a pattern that can be captured by a template, as the usage of the set of instances does not unify; 2. the pattern is already captured by a template, in which case we have found an instance of lack of reuse; otherwise 3. we have found one or more candidates (one for each non-unifiable usage of the instances of I) for new templates. The two first cases can be identified automatically, but the third needs user interaction to assess. First, a user should verify for each of the new templates that it is a meaningful pattern with respect to the domain; second, if the template is meaningful, a user must give the new template an appropriate name.

To remove the redundancy a candidate pair $\langle I, T \rangle$ represents, we can perform the following procedure for each template $t \in T$ and $T' = T \setminus \{t\}$. First we check for lack of reuse of t: this may only be the case if t's body has the same number of instances as there are templates in I. We verify the lack of reuse by checking if $t' \in T'$ contains t; this is done by verifying that t used as a query over t''s body yields an answer. If there is no lack of reuse, we can represent the instances of I as they are instantiated in t, as the body of a new template where all arguments are made into parameters. Again, we need to verify that the new template is contained in other templates in T' before we can refactor, and before any refactoring is carried out, a user should always assess the results.

Example 8. Applying the method for finding candidates to the library in Fig. 3(a), gives 19 candidate pairs, two of which are the 2nd and 3rd candidate pair of Ex. 7. The 1st dependency pair of Ex. 7 is not a candidate pair since the size of one of its elements ({BURGER}) is one.

By using the process of removing redundancies as described above, we will find that for the 2nd candidate pair of Ex. 7 we have a lack of reuse of SUBOBJECTALLVALUESFROM in BURGER, as discussed in previous examples. The two instances of SUBCLASSOF and OBJECTALLVALUESFROM in BURGER (see Ex. 5) can be therefore be replaced with the single instance: SUBOBJECTSOMEVALUESFROM(?Name, :hasCondiment, _:b3).

From the 3rd candidate pair in Ex. 7 there is no lack of reuse, but we can represent the pattern as the following template:

```
<NAME>(?x1, ?x2, ?x3, ?x4)
    :: (?x1, rdfs:label, ?x2), (?x1, skos:prefLabel, ?x3),
       (?x1, skos:definition, ?x4) .
```

The template and parameters should be given suitable names and parameters given a type, as exemplified by the ANNOTATION template (11) found in Fig. 3(b). The procedure of identifying dependency pairs and lack of reuse is implemented and demonstrated in the online walk-through at `ottr.xyz/event/2018-10-08-iswc/`.

For large knowledge bases, the set of candidate pairs might be very large, as it grows exponentially in the number of template instances in the worst case. This means that manually assessing all candidate pairs is not feasible, and smaller subsets of candidates must be automatically suggested. We have yet to develop proper heuristics for suggesting good candidates, but the cases with the the most common patterns (the candidates with largest T-sets), the largest patterns (the candidates with the largest I-sets), or large patterns that occur often could be likely sources for patterns to refactor. The latter of the three can be determined by maximising a weight-function, for instance of the form $f(\langle I, T \rangle) = w_1|I| + w_2|T|$. However, these weights might differ from use-case to use-case. Another approach for reducing the total number of candidates to a manageable size, is to let a user group some or all of the templates according to subdomain, and then only present candidates with instances fully contained in a single group. The idea behind such a restriction is that it seems likely that a pattern is contained within a subdomain. We give an example of these techniques in the following section.

5 Use Case Evaluation

In this section we outline an evaluation of OTTR templates in a real-world setting at the engineering company Aibel, and demonstrate in particular our process of finding and removing redundancies over a large, generated template library.

Aibel is a global engineering, procurement, and construction (EPC) service company based in Norway best known for its contracts for building and maintaining large offshore platforms for the oil and gas industry. When designing an offshore platform, the tasks of matching customer needs with partly overlapping standards and requirements as well as finding suitable products to match design specifications are highly non-trivial and laborious. This is made difficult by the fact that the source data is usually available only as semi-structured documents that require experience and detailed competence to interpret and assess. Aibel has taken significant steps to automate these tasks by leveraging reasoning and queries over their *Material Master Data* (MMD) ontology. It integrates this information in a modular large-scale ontology of ~200 modules and ~80,000 classes and allows Aibel to perform requirements analysis and matching with greater detail and precision and less effort than with their legacy systems. Since the MMD ontology is considered by Aibel as a highly valuable resource that gives them a competitive advantage, it is not publicly available.

The MMD ontology is produced from 705 spreadsheets prepared by ontology experts and populated by subject matter experts with limited knowledge of modelling and semantic technologies. The column headers of the spreadsheets specify how the data is to be converted into an ontology, and the translation is performed by a custom-built pipeline of

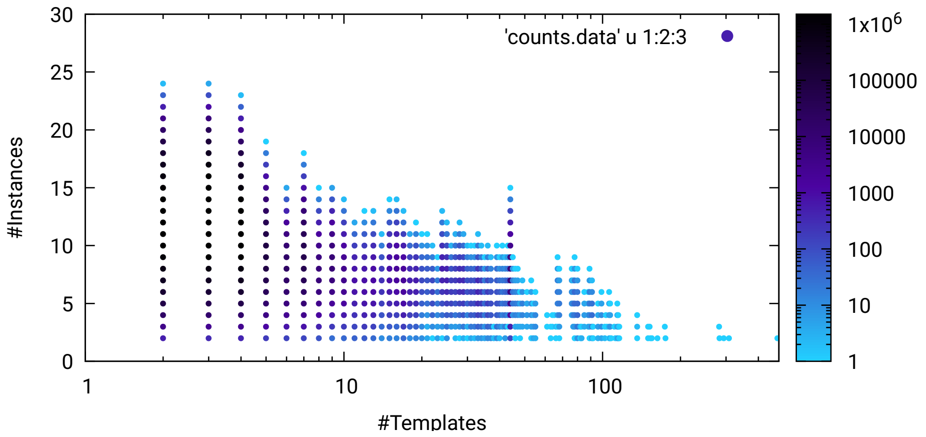

Fig. 4: A scatter plot of the sizes of the two sets for all candidate pairs from Aibel use case. The colour shade denotes the logarithm of the number of candidates at each point.

custom transformations, relational databases, and SPARQL CONSTRUCT transformations. The growing size and complexity of the system, the simple structure of the spreadsheets and lack of common modelling patterns make it hard to keep an overview of the information content of the spreadsheets and enforce consistent modelling across spreadsheets. The absence of overarching patterns also represents a barrier for Aibel's wish to extend the ontology to cover new engineering disciplines, as there are no patterns that are readily available for reuse.

The aim of our evaluation is to test whether OTTR templates and the tools presented in this paper can replace Aibel's current in-house built system and improve the construction and maintenance of the MMD ontology. By exploiting the simple structure of the spreadsheets we automatically generated OTTR templates: one for each spreadsheet (705 templates), one for each unique column header across spreadsheets (476 templates), and one for each axiom pattern used, e.g., existential restriction axiom (4 templates).

To analyse the large template library, we applied the algorithm for finding candidate pairs described in Sec. 4.2, giving a total of 54,795,593 candidate pairs. The scatter plot in Fig. 4 shows the distribution of sizes for the two sets; the largest number of instances and templates for a given candidate is 24 and 474, respectively. The large number of candidates makes it impossible to manually find potential templates, thus we employed the semi-automatic method described in the previous section to suggest possible improvements to the library. In order to demonstrate the process, we selected the candidates that contain a specific template, the template modelling a particular type of *pipe elbows* from the AMSE B16.9 standard, which is an often-used example from the MMD ontology. This template occurs in a total of 12,273 candidates. To reduce the number of candidates further, we removed candidates with instances of a generic character, such as `rdfs:label`, to end up with candidates with domain-specific templates. By using a weight function, we selected the candidate with the largest set of templates and at least 6 instances. From this candidate, with 33 templates and 7 instances, we obtained a template suggestion that we were able to verify is contained by all of the 33 templates in T, by using the template as a query over the templates in T. We added this new template to the library and refactored it into all the 33 templates using its pattern.

Fixing this single redundancy reduced the total number of candidates by over 1.8 million. This great reduction in candidates comes from the fact that fixing a redundancy represented by a candidate C can also fix the redundancies of candidates having a pattern that is contained in, contains, or overlaps C's pattern. This indicates that, despite a very large number of candidates, small fixes can dramatically reduce the overall redundancy. Furthermore, by automatically refactoring all lack of reuse in the entire library, the number of candidates decreases to under 3 million. The average number of instances per template went from 5.6 down to 2.7 after this refactoring. In addition to the redundancies fixed above, we were also able to detect equal templates (pairs of templates both having a lack of reuse of the other). Out of the 931 templates we analysed, only 564 were unique. Thus, we could remove a total of 367 redundant templates from the library. Note that all of the improvements made above should be reviewed by a user, as discussed in Sec. 4.2, to ensure that the new template hierarchy properly represents the domain.

The use case evaluation indicates that OTTR templates and tools can replace Aibel's custom built approach for transforming spreadsheets into ontologies. Indeed, OTTR greatly exceeds the expressivity of Aibel's spreadsheet structure and provides additional formal structure that can be used to analyse and improve the modelling patterns used to capture domain knowledge. As future evaluation, we want to work with Aibel's domain experts in order to identify promising heuristics for finding the best shared patterns. We believe that these new patterns and user requirements from Aibel may foster new ideas for added expressivity and functionality of OTTR languages and tools. Furthermore, we want to evaluate whether we can replace Aibel's hand-crafted queries with queries generated from templates. This would avoid the additional cost of maintaining a large query library, while benefiting from already existing templates and OTTR's compositional nature and tools for building and analysing the generated queries.

6 Related Work

Modularised ontologies, as well as the use and description of ontology design patterns, have attracted significant interest in recent years, as demonstrated by the multitude of languages and frameworks that have emerged. However, a hurdle for the practical large-scale use of ontology design patterns is the lack of a tool supported methodology; see [4] for a discussion of some of the challenges facing ontology design patterns. In this section we present selected work related to our approach that we believe represents the current state of the art.

An early account of the features, benefits and possible use-cases for a macro language for OWL can be found in [14].

The practical and theoretical aspects of OTTR templates were first introduced in [13] and [2]. This paper presents a more mature and usable framework, including formalisation and use of template relations, real-world evaluation, added expressivity in the form of optional parameters and expansion modes, and new serialisation formats.

GDOL [9] is an extension of the Distributed Ontology, Modelling, and Specification Language (DOL) that supports a parametrisation mechanism for ontologies. It is a metalanguage for combining theories from a wide range of logics under one formalism while supporting pattern definition, instantiation, and nesting. Thus it provides a broad formalism for defining ontology templates along similar lines as OTTR. To our knowledge, GDOL has yet to investigate issues such as dependencies and relationships between patterns, optional parameters, and pattern-as-query (the latter being listed as future work). A protege plugin for GDOL is in the works and DOL is supported by Ontohub (an online ontology and specification repository) and Hets (parsing and inference backend of DOL).

Ontology templates as defined in [1] are parameterised ontologies in $\mathcal{ALC}$ description logic. Only classes are parameterised, and parameter substitutions are restricted to class names. This is quite similar to our approach, yet it is not adapted to the semantic web, and nested templates and patterns-as-queries are not considered. Furthermore, it appears this project has been abandoned, as the developed software is no longer available.

OPPL [6] was originally developed as a language for manipulating OWL ontologies. Thus it supports functions for adding and removing patterns of OWL axioms to/from an ontology. It relies heavily on its foundations in OWL-DL and as such can only be used in the context of OWL ontologies. Despite this, the syntax of OPPL is distinct from that of RDF, thus requiring separate tools for viewing and editing such patterns, though a Protégé plugin does exist, in addition to a tool called *Populous* [7] which allows OPPL patterns to be instantiated via spreadsheets. By allowing patterns to return a single element (e.g., a class) OPPL supports a rather restricted form of pattern nesting as compared to OTTR.

Tawny OWL [10] introduces a Manchester-like syntax for writing ontology axioms from within the programming language Clojure, and allows abstractions and extensions to be written as normal Clojure code alongside the ontology. Thus the process of constructing an ontology is transformed into a form of programming, where existing tools for program development, such as versioning, testing frameworks, etc. can be used. The main difference from our approach is that Tawny OWL targets programmers and therefore tries to reuse as much of the standards and tools used in normal Clojure development, whereas we aim to reuse semantic technology standards and tools.

OPLa [5] is a proposal for a language to represent the relationships between ontologies, modules, patterns, and their respective parts. They introduce the OPLa ontology which describes these relationships with the help of OWL annotation properties. This approach does not, however, attempt to mitigate issues arising with the *use* of patterns, but focuses more on the description of patterns, than on practical use.

There are other tools and languages such as XDP [3], built on top of WebProtégé as a convenient tool for instantiating ODPs, the M^2 mapping language [11] that allows spreadsheet references to be used in ontology axiom patterns, and RDF shape languages, such as SHACL [8], that may be used to describe and validate patterns. Although these have similarities with OTTR, we consider these more specialised tools and languages, where for example analysis of patterns is beyond their scope.

7 Conclusion and Future Work

This paper presents OTTR, a language with supporting tools for representing, using and analysing ontology modelling patterns. OTTR has a firm theoretical and technological base that allows existing methods, languages and tools to be leveraged to obtain a powerful, yet practical instrument for ontology construction, use and maintenance.

For future work, the natural next step with respect to template library optimisation is to continue and expand the analysis of Section 4, both for existing and new template relations. In particular, it is natural to compare templates both syntactically using their full expansion and in terms of their semantic relationship. The latter would allow us, e.g., to answer questions about consistency and whether a given library is capable of describing a certain knowledge pattern. We also want to develop specialised editors for OTTR templates, such as a plugin for Protégé, and extend support for more input formats, such as accessing data from relational databases.

Acknowledgements. We would like to thank Per Øyvind Øverli from Aibel, and Christian M. Hansen from Acando for their help with the evaluation of OTTR. The second and fourth author were supported by Norwegian Research Council grant no. 230525.

References

1. M. Blasko, P. Kremen, and Z. Kouba. Ontology evolution using ontology templates. *Open Journal of Semantic Web (OJSW)*, 2:15–28, 01 2015.
2. H. Forssell et al. Reasonable macros for ontology construction and maintenance. In *Proc. of the 30th International Workshop on Description Logics*, 2017.
3. K. Hammar. Ontology Design Patterns in WebProtege. In *Proc. of the ISWC 2015 Posters & Demonstrations Track*, 2015.
4. K. Hammar et al. Collected research questions concerning ontology design patterns. In P. Hitzler et al., editors, *Ontology Engineering with Ontology Design Patterns*, pages 189–198. IOS Press, 2016.
5. P. Hitzler et al. Towards a simple but useful ontology design pattern representation language. In *Proc. of the 8th Workshop on Ontology Design and Patterns*, 2017.
6. L. Iannone, A. L. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009.

7. S. Jupp et al. Populous: a tool for building OWL ontologies from templates. *BMC Bioinformatics*, 13(S-1):S5, 2012.
8. H. Knublauch and D. Kontokostas. Shapes constraint language (SHACL), 2017. W3C recommendation.
9. B. Krieg-Brückner and T. Mossakowski. Generic ontologies and generic ontology design patterns. In *Proc. of the 8th Workshop on Ontology Design and Patterns*, 2017.
10. P. Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. In *OWLED*, 2013.
11. M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. M2: A Language for Mapping Spreadsheets to OWL. In *OWLED*, 2010.
12. C. K. Ogden and I. A. Richards. *The Meaning of Meaning*. Harvest Book, 1946.
13. M. G. Skjæveland et al. Pattern-based ontology design and instantiation with reasonable ontology templates. In *Proc. of the 8th Workshop on Ontology Design and Patterns*, 2017.
14. D. Vrandečić. Explicit knowledge engineering patterns with macros. In *Proc. of the Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, 2005.

Paper 5

Making a Case for Formal Relations over Ontology Patterns

Daniel P. Lupp, Leif Harald Karlsen, and Martin G. Skjæveland. “Making a Case for Formal Relations over Ontology Patterns.” In: *Proceedings of the 9th Workshop on Ontology Design and Patterns (WOP 2018) co-located with the 17th International Semantic Web Conference (ISWC 2018), Monterey, CA, October 9, 2018.* Vol. 2195. CEUR Workshop Proceedings. CEUR-WS.org, 2018 [P5]

Making a Case for Formal Relations over Ontology Patterns

Daniel P. Lupp, Leif Harald Karlsen, and Martin G. Skjæveland

Department of Informatics, University of Oslo
`{danielup,leifhka,martige}@ifi.uio.no`

Abstract. There have recently been multiple frameworks proposed to formalize the definition and instantiation of recurring patterns for ontology construction and maintenance. Such formal frameworks can also provide the means necessary for discussing how such patterns can be related to one another, both syntactically and semantically. This has the potential for organizing pattern libraries, robust handling of maintenance tasks, such as redundancy removal, and defining heuristics for what constitutes a "good" pattern. This short paper aims to provide a common ground for discussions on formal relations between ontology patterns. We discuss interesting relations with motivating examples as well as state open questions concerning relations for optimizing the creation, instantiation, and maintenance of ontology patterns.

1 Templating Framework for Ontology Patterns

Recently, multiple frameworks have been proposed for the creation and use of ontology patterns [3,2]. This more formal approach enables the study of how patterns are related to one another in ways that permit automatic analysis and repair. In order to adequately discuss formally defined relationships between patterns, we employ the notion of a templating mechanism for patterns using the following generic definitions: An *(ontology) pattern* is a set of OWL axioms or RDF triples; a *templating framework* has the following characteristics, where we consider the first three mandatory and the remaining optional:

1. identifiable patterns, called *templates*;
2. declaration of fixed and variable template parameters;
3. precise instantiation of templates;
4. support for nested template definitions, i.e., templates defined using other templates;
5. typed parameters;
6. cardinality for parameters, e.g,. *optional* and *mandatory*;
7. inherited semantics of the underlying language.

$$\overbrace{\underbrace{\textsc{SubClassOf}}_{\text{name}}(\underbrace{\text{?sub : 1 class, ?super : 1 class}}_{\text{parameters with type and cardinality}})}^{\textit{head}\text{, identifying template and parameters}} :: \overbrace{\{\underbrace{\textsc{Triple}(\text{?sub, rdfs:subClassOf, ?super})}_{\text{instance with fixed and variable arguments}}\}}^{\textit{body}\text{, representing the pattern}} .$$

Fig. 1: Reasonable Ontology Templates (OTTR) exemplified.

The discussion is motivated by our work with *Reasonable Ontology Templates* (OTTR) [3,1] which implements these features. Figure 1 gives a schematic example of an OTTR template, for more details we refer to [3]. The OTTR template framework has been successfully verified for construction and maintenance tasks on Aibel's large-scale Material Master Data (MMD) ontology. There approximately 1000 templates were used to represent the spreadsheet formats created and populated by the project to capture the domain knowledge for generating an ontology of ca. 80,000 classes. Experimental analysis of these templates based on simple relations has revealed a considerable potential for optimization of their design. We believe that a richer set of template relations is both possible and required in order to gain a more fine-grained and effective characterization of templates and template libraries.

In this paper we discuss the benefit and challenges of defining formal relations between ontology templates and identify future directions of research in this area, with an agnostic perspective as to which templating formalism is used. As such, the intention is for the reader to quickly be able to translate the discussion and examples given into the formalism of their choosing. Section 2 shows how simple relations may be used to identify redundancies in a template library. Section 3 presents the possibilities and potential for defining new relations over ontology templates.

2 Simple and Useful Template Relations

Using only the first four characteristics of a template framework given in Section 1, we can define some basic relationships between templates:

directly depends S is said to *directly depend on* T if S contains an instance of T in its definition.

depends *depends* is the transitive closure of *directly depends*.

dependency-overlaps S *dependency-overlaps* T if there exists a template upon which both S and T directly depend.

Burger1(?Name : 1 class, ?Condiments : + class)
:: SubClassOf(?Name, :Burger), ObjectUnionOf(_:b3, ?Condiments),
SubClassOf(?Name, _:b2), ObjectAllValuesFrom(_:b2, :hasCondiment, _:b3). (1)

SubObjectAllValuesFrom(?x : 1 class, ?Property : 1 objectProperty, ?Range : 1 class)
:: SubClassOf(?x, _:b1), ObjectAllValuesFrom(_:b1, ?Property, ?Range) . (2)

(a) Example of lack of reuse.

Burger2(?Name : 1 class, ?Condiments : + class)
:: SubClassOf(?Name, :Burger), ObjectUnionOf(_:b1, ?Condiments), (3)
SubObjectAllValuesFrom(?Name, :hasCondiment, _:b1). (4)

Pizza(?Name : 1 class, ?Toppings : + class, ?Country : ? individual)
:: SubClassOf(?Name, :NamedPizza), ObjectUnionOf(_:b1, ?Toppings), (5)
SubObjectAllValuesFrom(?Name, :hasTopping, _:b1), (6)
SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country).

(b) Example of uncaptured pattern.

Food1(?Name : 1 class, ?Type : 1 class, ?Ex : + class, ?hasEx : 1 objProp)
:: SubClassOf(?Name, ?:Type), ObjectUnionOf(_:b1, ?Ex),
SubObjectAllValuesFrom(?Name, ?:hasEx, _:b1).

(c) Captured pattern.

Food2(?Name : 1 class, ?Type : 1 class, ?Ex : + class, ?hasEx : 1 objProp, ?Country : ? individual)
:: SubClassOf(?Name, ?:Type), ObjectUnionOf(_:b1, ?Ex),
SubObjectAllValuesFrom(?Name, ?:hasEx, _:b1),
SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country).

(d) Captured pattern with optionals.

Fig. 2: Templates demonstrating different redundancies and suggested solutions.

overlaps S *overlaps* T if there exist template instances i_S, i_T in the definition of S and T and substitutions ρ and η of the parameters of S and T resp. such that $\rho(i_S) = i_T$ and $\eta(i_T) = i_S$.

contains S *contains* T if there exists a substitution ρ of the parameters of T such that ρ applied to the pattern of T is a subset of the pattern of S.

These relations can be used to identify redundancies in a set of templates. In [3], two types of redundancies are considered: *lack of reuse* and *uncaptured pattern*. *Lack of reuse* occurs when a template duplicates the pattern of another template rather than instantiating it. This is exemplified by the two templates in Figure 2(a); BURGER1 *contains* SUBOBJECTALLVALUESFROM (see (1) and (2)). The template BURGER2 in Figure 2(b) is the result of fixing this redundancy (replacing (1) with (4)). *Uncaptured pattern* is the case when multiple templates make use of a pattern which is not represented by a template. This is illustrated by the two templates of Figure 2(b); these templates dependency-overlap, as seen in (3)–(4) and (5)–(6). This is not an occurrence of lack of reuse, as PIZZA cannot instantiate BURGER2 given they both use different fixed parameter resources (e.g., hasCondiments (4) and hasTopping (6)). The uncaptured pattern may be refactored out as a separate template; this is implemented by FOOD1 in Figure 2(c).

In the analysis of the $\sim$1000 templates used for Aibel's MMD ontology, ca. 55 million potential redundancies were identified and 367 possibly superfluous templates found. When deciding how to refactor the template library we used a manual approach, targeting large templates that model specific domain facts. Fixing a single redundancy reduced the total number of potential redundancies by more than 1.8 million. Although the analysis is clearly useful and shows promising results, we believe these large figures show that one has yet to find precise, effective means to characterize and repair template libraries.

3 Defining New Ontology Template Relations

The relations defined in the previous section use only a few of the basic characteristics for a templating mechanism given in Section 1. The following is a non-exhaustive list of building blocks for defining formal relations between templates using both the characteristics of a templating framework and of the underlying language. The list is presented along with a brief summary of what kind of functionality each provides and typical examples of possible relations defined using these characteristics:

Syntax and semantics of framework These allow syntactic and semantic relations such as containment, entailment, and consistency, e.g., two templates could be said to be *inconsistent* if for any substitutions of their parameters the use of both in conjunction yields an inconsistent ontology.

Parameters This allows for relations that take into consideration the number of parameters, their types, and whether they are optional or not. For instance, one could say that T *strengthens the typing* of S if T directly depends on S and additionally requires a more specific type for at least one of its parameters.

Expansion This allows for relations over both unexpanded and expanded templates. For example, *expanded containment* could be defined to hold between T and S if the expansion of T contains the expansion of S.

Metrics Characteristics such as the template's *arity* (the number of parameters), *width* (the size of its pattern) and *depth* (the height of its expansion tree).

Syntax and semantics of used vocabulary It is useful to distinguish between templates that are completely generic (all parameters are variable) and those that use a specific vocabulary, be it a "logical" vocabulary like RDF or OWL or a more special purpose vocabulary. As for semantic relations such as consistency, these can be set to consider or ignore the semantics of "external" ontologies.

These building blocks give many possibilities for characterizing templates and template libraries. We are particularly interested in identifying relations that are useful for structuring template libraries for maintenance tasks, such as redundancy removal, and that make it easier for the user to find the relevant template for the modeling task at hand. In addition, finding heuristics with which libraries can be optimized according to specific metrics, which may vary from case to case, could be a valuable asset. Ideally, the relations and heuristics should be efficiently computable and be easily understood by the user. Finding these requires more theoretical and empirical study. We conclude this section by demonstrating some of the challenges of such studies.

When defining relations using the above building blocks, one should proceed with caution as some of them are incredibly powerful. For instance, allowing parameters to be optional provides more possibilities for discovering candidates for uncaptured patterns, as demonstrated in Figure 2: Consider the PIZZA and BURGER2 templates from Figure 2(b). Without allowing optional parameters, a suitable template that describes the uncaptured pattern can be seen in Figure 2(c). With optionals, however, the BURGER2 and PIZZA templates can be generalized into a single template by declaring ?Country as an optional parameter, seen in Figure 2(d). Optionals thus provide a lot of useful functionality; however, in general they

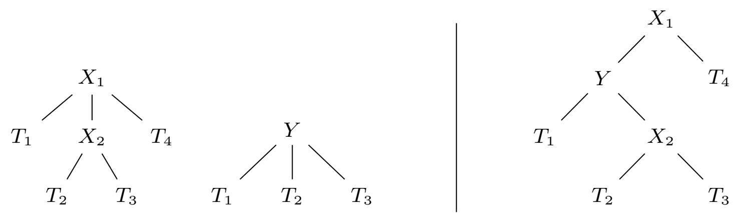

Fig. 3: A complex case of lack of reuse. Templates are schematically represented by trees where an edge represents a direct dependency. On the left, there is a lack of reuse between the templates X_2 and Y, which, if fixed (by replacing T_2 and T_3 in Y with X_2), introduces a new lack of reuse between Y and X_1. Fixing also this redundancy (replacing T_1 and X_2 in X_1 with Y) results in the configuration of templates seen on the right.

allow for any set of templates to be summarized into a single template. As this is likely counter-productive in most circumstances, it would be important to identify heuristics for when optionals should or should not be used when defining templates for uncaptured patterns.

Other complex relationships can occur during the fixing of lack of reuse or uncaptured pattern. An example that illustrates this is shown in Figure 3, where fixing one instance of lack of reuse in Y introduces a new lack of reuse in X_1. This indicates a more complex form of lack of reuse between X_1 and Y in the original library, but also, more generally, that repairing redundancies is an iterative process. This type of composition of relations opens the door to new types of complex relationships. Furthermore, if the template definitions have type and cardinality restrictions set on the parameters, finding possible optimizations or redundancies becomes even more involved: for instance, if parameter types are too strict then the possible reuse of these templates is reduced.

4 Conclusion and Future Work

In this paper, we introduce an abstract templating framework and the building blocks with which formal relations over ontology patterns and templates may be defined. We give examples of formal relations that have a demonstrable benefit to template library maintenance (by removing redundancy) as well as provide useful functionality for tools geared towards template creation, documentation, and discovery.

The framework, relations and building blocks described in this paper serve primarily as a basis for discussion. We believe it important to identify (1) what functionality we desire for improved usability and in maintenance and creation tools, (2) what relations these require, and (3) what properties aside from the aforementioned building blocks a templating framework should support.

References

1. H. Forssell, D. P. Lupp, M. G. Skjæveland, and E. Thorstensen. Reasonable macros for ontology construction and maintenance. In *Proc. of 30th DL Workshop*, 2017.
2. B. Krieg-Brückner and T. Mossakowski. Generic ontologies and generic ontology design patterns. In *Proc. of the 8th Workshop on Ontology Design and Patterns*, 2017.
3. M. G. Skjæveland, D. P. Lupp, L. H. Karlsen, and H. Forssell. Practical ontology pattern instantiation, discovery, and maintanence with reasonable ontology templates. Accepted for ISWC 2018 research track, 2018.

Paper 6

Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity

Henrik Forssell, Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen. "Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity." In: *CoRR* abs/1809.10436 (2018). arXiv: `1809.10436`. URL: `http://arxiv.org/abs/1809.10436` [P6]

Generating Ontologies from Templates: A Rule-Based Approach for Capturing Regularity

Henrik Forssell Christian Kindermann Daniel P. Lupp
Uli Sattler Evgenij Thorstensen

Technical Report

Contents

Abstract

We present a second-order language that can be used to succinctly specify ontologies in a consistent and transparent manner. This language is based on ontology templates (OTTR), a framework for capturing recurring patterns of axioms in ontological modelling. The language, and our results are independent of any specific DL.

We define the language and its semantics, including the case of negation-as-failure, investigate reasoning over ontologies specified using our language, and show results about the decidability of useful reasoning tasks about the language itself. We also state and discuss some open problems that we believe to be of interest.

1 Introduction

The phenomenon of frequently occurring structures in ontologies engineering (OE) has received attention from a variety of angles. One of the first accounts is given in [6], where repeated versions of general conceptual models are identified. Similar observations gave rise to the notion of *Ontology Design Patterns* (ODP) as abstract descriptions of best practices in OE [15, 4, 22]. Another view, emphasizing common ontological distinctions, led to the emergence of *Upper Ontologies* which aim to categorize general ideas shareable across different domains [16]. Orthogonal to such conceptual patterns, the existence of syntactic regularities in ontologies has been noted and some aspects of their nature have been analyzed [30, 29, 31].

In this paper, we propose a new language that allows expressing patterns of repeated structures in ontologies. This language is rule-based and has both a model-theoretic and a fixpoint semantics, for which we show that they coincide. In contrast to other rule languages "on top of" DLs, in this language, firing a rule results in the addition of TBox and/or ABox axioms, with the goal to succinctly describe ontologies, thereby making them more readable and maintainable.

Given that DL ontologies are sets of axioms, an ontology provides no means to arrange its axioms in a convenient manner for ontology engineers. In particular, it is not possible to group conceptually related axioms or indicate interdependencies between axioms. While ontology editors such as Protégé[1] display an ontology through a hierarchy of its entities, conceptual interdependencies between axioms are hidden and the underlying structural design of an ontology remains obfuscated.

[1]https://protege.stanford.edu/

Example 1.1 Consider the ontology

$$\begin{aligned} \mathcal{O}_1 = \{\mathsf{Jaguar} \sqsubseteq \mathsf{Animal}, &\qquad \mathsf{Jaguar} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Jaguar}, &\quad (1)\\ \mathsf{Tiger} \sqsubseteq \mathsf{Animal}, &\qquad \mathsf{Tiger} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Tiger}, &\quad (2)\\ \mathsf{Lion} \sqsubseteq \mathsf{Animal}, &\qquad \mathsf{Lion} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Lion}\} &\quad (3) \end{aligned}$$

Then, an ontology editor will group the entities Jaguar, Tiger and Lion under Animal according to their class hierarchy.

However, $\mathcal{O}_1$ contains no indication that every subclass X of Animal can have only children of the same class X. Assume this regularity is no coincidence but a desired pattern that should hold for any subclass of Animal. Currently, ontology engineers have no means of expressing or enforcing such a pattern other than dealing with the ontology as a whole, inspecting all axioms separately, and making necessary changes manually. □

Expressing patterns such as in Example 1.1 explicitly has a potential to reveal some aspects of the intentions for the design of an ontology.

Example 1.2 Consider the ontology

$$\mathcal{O}_2 = \{\mathsf{Jaguar} \sqsubseteq \mathsf{Animal},\ \mathsf{Tiger} \sqsubseteq \mathsf{Animal},\ \mathsf{Lion} \sqsubseteq \mathsf{Animal}\}$$ □

In addition, consider the rule

$$g\colon \underbrace{\{?X \sqsubseteq \mathsf{Animal}\}}_{\text{Body}} \rightarrow \underbrace{\{?X \sqsubseteq \forall \mathsf{hasChild}.?X\}}_{\text{Head}},$$

where $?X$ is a variable. We can interpret the body of this rule as a query which, when evaluated over the ontology $\mathcal{O}_2$, returns substitutions for $?X$. These substitutions can then be used to instantiate the axioms in the head of the rule. Firing the above rule over $\mathcal{O}_2$ would add all those resulting axioms to $\mathcal{O}_2$, thereby reconstructing $\mathcal{O}_1$ from Example 1.1.

In the following, we will call such rules *generators*. The possible benefits of generators are threefold. Firstly, $\mathcal{O}_2$ in combination with g is easier to understand because g makes a statement about all subconcepts of Animal that *the type of an animal determines the type of its children*. This is a kind of meta-statement about concepts which a user of an ontology can usually only learn by inspecting (many) axioms in an ontology. Secondly, $\mathcal{O}_2$ in combination with g is easier to maintain and extend compared to $\mathcal{O}_1$, where a user would have to manually ensure that the meta-statement continues to be satisfied after new concepts have been added. Thirdly, conceptual relationships captured in a generator such as g are easy to reuse and can foster interoperability between ontologies in the spirit of ontology design patterns.

We close this section with more elaborate examples to demonstrate the benefits generators such as g can provide.

1.1 Examples

Example 1.3 (Composition) Assume we want to model typical roles in groups of social predatory animals. One such a role would be that of a hunter. A challenge for representing such knowledge is that different collective nouns are used for different animals, e.g. a group of lions is called a "pride", a group of wild dogs is called a "pack", a group of killer whales is called a "pod", etc. Therefore, a mechanism that can conveniently iterate over all these group formations would be beneficial.

Consider the following query Q_1:

$$\begin{aligned} Q_1 = \{?X &\sqsubseteq \mathsf{Animal}, && (4)\\ ?X &\sqsubseteq \exists \mathsf{eats}.\mathsf{Animal}, && (5)\\ ?X &\sqsubseteq \exists \mathsf{hunts}.\mathsf{Animal}, && (6)\\ ?Y &\sqsubseteq \mathsf{SocialGroup}, && (7)\\ ?X &\sqsubseteq \exists \mathsf{socialisesIn}.?Y, && (8)\\ ?Y &\sqsubseteq \exists \mathsf{hasMember}.?X, && (9)\\ \mathsf{socialisesIn} &\equiv \mathsf{hasMember}^{-}\} && (10)\end{aligned}$$

□

Lines 4–6 bind the variable $?X$ to a predatory animal. Line 7 binds the variable $?Y$ to a type of social group and lines 8–10 associate a particular type of animal with its respective social group. Given the bindings for $?X$ and $?Y$ it is straightforward to express that a particular type of predator $?X$ is a hunter in its respective social group, namely: $?Y \sqsubseteq \exists \mathsf{hasHunter}.?X$. A generator such as in Example 1.2 could capture this relationship:

$$g_1 : Q_1 \rightarrow \{?Y \sqsubseteq \exists \mathsf{hasHunter}.?X\}$$

Example 1.4 (Extension) Extending generator g_1 from Example 1.3 to capture more specialised knowledge is straightforward. Consider predatory ants of the family Formicidae. These ants generally live in colonies with an elaborate social organisation consisting of workers, drones, queens, etc.

First, we extend query Q_1 with the following axioms:

$$\begin{aligned} Q_2 = Q_1 \cup \{?X &\sqsubseteq \mathsf{Formicidae}, && (11)\\ ?Z &\sqsubseteq ?Y && (12)\\ ?X &\sqsubseteq \exists \mathsf{socialisesIn}.?Z\} && (13)\end{aligned}$$

Axiom 11 requires $?X$ to bind to a type of Formicidae, e.g. $?X = \mathsf{SafariAnt}$. According to query Q_1, the variable $?Y$ binds to a general SocialGroup, e.g. $?Y = \mathsf{AntColony}$. Then, axiom 12 binds $?Z$ to a more specialised subgroup of a $?Y$. Finally, axiom 13 ensures that this subgroup $?Z$ is associated with $?X$. So for $?X = \mathsf{SafariAnt}$ we get $?Z = \mathsf{SafariAntColony}$.

Next, we can specify the generator to add all desired axioms based on matches of query Q_2 specialised for ants:

$$\begin{aligned} &g_2 : Q_2 \rightarrow \\ &\quad \{?Z \sqsubseteq \exists \mathsf{hasHunter}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasWorker}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasDrone}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasQueen}.?X\} \end{aligned}$$

Note how the body and head of generator g_1 from Example 1.3 have been reused and extended only by set unions. □

Example 1.5 (Negative Guards) Often, general relationships are subject to exceptions. While most ants hunt and feed cooperatively, there are some genera of ants, e.g. Myrmecia, that do not. Therefore, g_2 in Example 1.4 would generate an undesired axiom, namely

$$\mathsf{MyrmeciaAntColony} \sqsubseteq \mathsf{hasHunter}.\mathsf{MyrmeciaAnt}$$

. This motivates guards in the body of generators that may not only specify positive constraints but also negative ones:

$$\begin{aligned} Q_3 = Q_2 \cup \{&\textbf{not}\ \ ?X \sqsubseteq \mathsf{MyrmeciaAnt}, \\ &\textbf{not}\ \ ?Z \sqsubseteq \mathsf{MyrmeciaAntColony}\} \end{aligned}$$

$$\begin{aligned} &g_3 : Q_3 \rightarrow \\ &\quad \{?Z \sqsubseteq \exists \mathsf{hasHunter}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasWorker}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasDrone}.?X, \\ &\quad\ ?Z \sqsubseteq \exists \mathsf{hasQueen}.?X\} \end{aligned}$$

One might argue that the effect of negative guards could also be achieved by positive guards using negated concepts in DL, i.e. $?X \sqsubseteq \neg\mathsf{MyrmeciaAnt}$ instead of **not**$?X \sqsubseteq \mathsf{MyrmeciaAnt}$. However, this approach would necessitate the introduction of a potentially large number of axioms of type $?X \sqsubseteq \neg\mathsf{MyrmeciaAnt}$ in the given ontology. This can be avoided by using g_3.

Another advantage of negative guards is the possibility to explicitly express default assumptions for lack of better knowledge. An ant colony of a certain genus usually consists of only ants of this genus, e.g.

$$\mathsf{SafariAntColony} \sqsubseteq \forall \mathsf{hasMember}.\mathsf{SafariAnt}. \tag{14}$$

However, some genera of ants are social parasites that enslave other ant species. In such a case, the default assumption about the homogeneity of an ant colony is wrong and the axiom 14 should not be added.

$$\begin{aligned} Q_4 = \{ & ?X \sqsubseteq \mathsf{Ant}, \\ & ?Y \sqsubseteq \mathsf{AntColony}, \\ & ?Y \sqsubseteq \exists \mathsf{hasMember}.?X, \\ & ?Z \sqsubseteq \mathsf{Ant}, \\ & ?X \sqsubseteq \neg ?Z, \\ \textbf{not}\ & ?X \sqsubseteq \exists \mathsf{enslaves}.?Z, \\ \textbf{not}\ & ?Y \sqsubseteq \exists \mathsf{hasMember}.?Z, \} \end{aligned}$$

$$g_4 \colon Q_4 \to \{?Y \sqsubseteq \forall \mathsf{hasMember}.?X\}$$ □

Example 1.6 (Recursion) Contagious diseases may be transmitted between animals sharing a habitat. Overlapping habitats of infected animals may result in a propagation of diseases across habitats.

Assume there is an overlap between habitats H_1, H_2, H_3 such that there is no overlap between H_1 and H_3, H_{12} describes the overlap between H_1 and H_2, and H_{23} describes the overlap between H_2 and H_3 (see Figure 1). Then, a disease infected animal living in H_1 may affect an animal in H_2 which in turn may affect an animal in H_3. Such an iterative process may be captured by repeatedly applying a single generator.

Consider the following query:

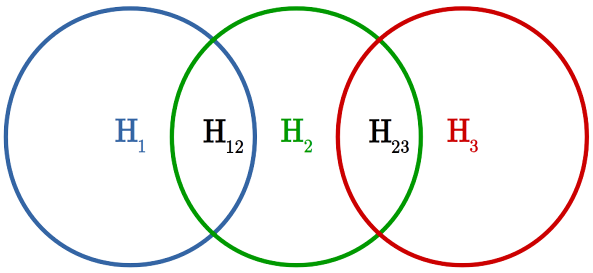

Figure 1: Overlapping Habitats

$$\begin{aligned}
Q_5 = \{?X &\sqsubseteq \mathsf{Animal}, && (15)\\
?Y &\sqsubseteq \mathsf{Animal}, && (16)\\
?D &\sqsubseteq \mathsf{ContagiousDisease}, && (17)\\
?H &\sqsubseteq \mathsf{Habitat}, && (18)\\
?X &\sqsubseteq \exists \mathsf{suffersFrom}.?D, && (19)\\
?Y &\sqsubseteq \forall \mathsf{isSusceptibleTo}.?D, && (20)\\
?X &\sqsubseteq \exists \mathsf{livesIn}.?H, && (21)\\
?Y &\sqsubseteq \exists \mathsf{livesIn}.?H\} && (22)
\end{aligned}$$

Axioms 19 and 20 express the requirements for a disease to be transmitted between animals while axioms 21 and 22 capture the requirement of a shared environment. Using query Q_5, we can represent the propagation of a disease between animals across habitats:

$$g_5 : Q_5 \rightarrow \{?Y \sqsubseteq \exists \mathsf{suffersFrom}.?D\} \qquad \square$$

Clearly, the generation of an instance of $?Y \sqsubseteq \exists \mathsf{suffersFrom}.?D$ could yield a new match for Q_5 in the body of g_5. Therefore, generator g_5 has to be applied repeatedly until a fixpoint is reached.

Example 1.7 (Encapsulation) Inspecting the queries Q_1, Q_2, and Q_3 in Examples 1.3–1.5, it is apparent that different parts in the queries correspond to different conceptual ideas. For example, in query Q_1 the axioms can be grouped into ones about predators and others about social groups. Such a grouping would provide valuable information for an ontology engineer to indicate conceptual relationships between certain sets of axioms:

$$\left.\begin{aligned}\{?X &\sqsubseteq \mathsf{Animal},\\ ?X &\sqsubseteq \exists \mathsf{eats}.\mathsf{Animal},\\ ?X &\sqsubseteq \exists \mathsf{hunts}.\mathsf{Animal}\}\end{aligned}\right\}\text{Predator}$$

$$\left.\begin{aligned}\{?Y &\sqsubseteq \mathsf{SocialGroup},\\ ?X &\sqsubseteq \exists \mathsf{socialisesIn}.?Y,\\ ?Y &\sqsubseteq \exists \mathsf{hasMember}.?X,\\ \mathsf{socialisesIn} &\equiv \mathsf{hasMember}^{-}\}\end{aligned}\right\}\text{Social Group}$$

Reasonable ontology templates [38, 14], OTTR for short, introduced a framework for indicating such conceptual relationships. A template is defined as a named ontology with a set of variables. The variables can be instantiated with concept and role expressions to yield a set of valid axioms. Moreover, templates may be composed to give rise to more complex templates. Choosing intention-revealing names for templates and composing appropriately named templates may improve ontology comprehension by making the structural design of an ontology visible.

A template, i.e. a set of axioms with variables, can also be interpreted as a query, asking for concept and role expressions in an existing ontology that match the pattern represented by the template. These expressions can then, in principle, be fed into a different template to produce new axioms. This idea captures conceptual interdependencies between templates or, more generally, axiomatic patterns.

Clearly, it is straightforward to integrate OTTR as part of a preprocessing step into our rule language. This has not only the potential to foster the reuse of conceptually related set of axioms in an intention-revealing manner, but can also to further improve the maintainability of generators by the principle of information hiding. A change in a template will be propagated automatically to all instances of the use of the template. □

2 Preliminaries

Let N_I, N_C, and N_R be sets of *individual*, *concept*, and *role names*, each containing a distinguished subset of *individual*, *concept*, and *role variables* V_I, V_C, and V_R. A *concept* (resp. *role*) is either a concept name (resp. role name) or a concept expression (resp. role expression) built using the usual DL constructors [2]. Since we do not distinguish between TBoxes and ABoxes, an *axiom* is either an assertion of the form $\mathsf{C}(\mathsf{a})$ or $\mathsf{R}(\mathsf{a},\mathsf{b})$ for a concept C, role R, and individual names a,b or an inclusion statement $\mathsf{C} \sqsubseteq \mathsf{D}$ for concepts or roles C and D. A *theory* is a (possibly infinite) set of axioms, whereas an *ontology* is a finite set of axioms. A set $\mathcal{L}$ of individuals, concepts, and roles is called a *language*.

A *template* T is an ontology, and we write $T(V)$ for $V \subseteq V_I \cup V_C \cup V_R$ the set of variables occurring in T. For the sake of brevity, we occasionally omit the variable set V when it is either clear from context or nonvital to the discussion. Templates can be *instantiated* by applying a substitution to them. A *substitution* σ is a function that maps individual, concept, and role variables to individuals, concepts, and roles respectively. We require that substitutions respect the type of a variable, so that the result of instantiating a template is a well-formed ontology. For $\mathcal{L}$ a language, an *$\mathcal{L}$-substitution* is one whose range is a subset of $\mathcal{L}$. The *$\mathcal{L}$-evaluation of T over $\mathcal{O}$*, written $\mathsf{eval}(T,\mathcal{O},\mathcal{L})$, is the set of substitutions defined as follows:

$$\mathsf{eval}(T,\mathcal{O},\mathcal{L}) = \{\sigma \text{ an } \mathcal{L}\text{-substitution} \mid \mathcal{O} \models T\sigma\},$$

where $T\sigma$ is the instantiation of T with σ. Furthermore, we define $\mathsf{eval}(\emptyset,\mathcal{O},\mathcal{L})$ to be the set of all $\mathcal{L}$-substitutions.

Finally, we say that an ontology *$\mathcal{O}$ is weaker than $\mathcal{O}'$* if $\mathcal{O}' \models \mathcal{O}$, and *strictly weaker* if the reverse does not hold.

3 Generators and GBoxes

In this section we define the syntax and semantics of generators and GBoxes and discuss some examples.

Definition 3.1 A *generator* g is an expression of the form $T_B(V_B) \to T_H(V_H)$, for $T_B(V_B), T_H(V_H)$ templates with $V_H \subseteq V_B$. T_B and T_H are respectively called the *body* and *head* of g, and we write $B(g)$ and $H(g)$ to denote them. □

Example 3.2 $g\colon \{?X \sqsubseteq \mathsf{Animal}\} \to \{?X \sqsubseteq \forall \mathsf{hasChild}.?X\}$ is a generator, with a single variable $?X$. □

Next, we define the semantics for generators and sets of generators based on entailment to ensure that generators behave independent of the syntactic form of an ontology. In this choice we diverge from the work done on OTTR [38], as OTTR template semantics is defined syntactically.

Definition 3.3 Let $g\colon T_B(V_B) \rightarrow T_H(V_H)$ be a generator. A theory $\mathcal{O}$ *satisfies* g *wrt.* $\mathcal{L}$ if, for every $\mathcal{L}$-substitution σ such that $\mathcal{O} \models T_B\sigma$, we have $\mathcal{O} \models T_H\sigma$. $\square$

Example 3.4 Consider the generator g from Example 3.2. The theory $\mathcal{O}_1 = \{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}, \mathsf{Turtle} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Turtle}, \mathsf{Mammal} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Mammal}\}$ satisfies g, while the theory $\mathcal{O}_2 = \{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}\}$ does not. $\square$

A set G of generators is called a *GBox*. Furthermore, we define the set $B(G)$ (resp. $H(G)$) as the set of all bodies (resp. heads) occurring in G, i.e., they are sets of ontologies.

Definition 3.5 Let G be a GBox, $\mathcal{O}$ an ontology, and $\mathcal{L}$ a language. The *expansion of* $\mathcal{O}$ *and* G *in* $\mathcal{L}$, written $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$, is the smallest set of theories $\mathcal{O}'$ such that

(1) $\mathcal{O}' \models \mathcal{O}$,

(2) $\mathcal{O}'$ satisfies every $g \in G$ w.r.t. $\mathcal{L}$, and

(3) $\mathcal{O}'$ is entailment-minimal, i.e. there is no $\mathcal{O}''$ strictly weaker than $\mathcal{O}'$ satisfying (1) and (2). $\square$

We call the theories in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ *expansions*. This definition corresponds to the model-theoretic Datalog semantics, with consequence rather than set inclusion. Since axioms can be rewritten to be subset-incomparable, entailment-minimality is used rather than subset minimality. For example, consider $\{A \sqsubseteq B, B \sqsubseteq C\}$ and $\{A \sqsubseteq C\}$: the second one is not a subset of the first one, but weaker than it.

Example 3.6 Recall the generator g from Example 3.2, and let G be a GBox consisting of g alone. Let $\mathcal{O} = \{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}\}$, and let $\mathcal{L}$ be the set of all concept names. Then $\{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}, \mathsf{Turtle} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Turtle}, \mathsf{Mammal} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Mammal}\} \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. $\square$

4 Results

We show that the semantics defined in the previous section coincides with a fixpoint-based one, investigate the role played by the language $\mathcal{L}$, and investigate generators with negated templates.

Theorem 4.1 *For every G, $\mathcal{O}$, and $\mathcal{L}$, we have that any two $\mathcal{O}_1, \mathcal{O}_2 \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ are logically equivalent.* □

Proof Assume for contradiction that this is not the case. Then there exist $\mathcal{O}_1, \mathcal{O}_2 \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ such that $\mathcal{O}_1 \not\models \mathcal{O}_2 \not\models \mathcal{O}_1$ because otherwise, one would be strictly weaker than the other, contradicting the definition of $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. In particular, there exist α and β such that:

$$\mathcal{O}_1 \models \alpha, \qquad \mathcal{O}_2 \not\models \alpha \tag{23}$$
$$\mathcal{O}_2 \models \beta, \qquad \mathcal{O}_1 \not\models \beta \tag{24}$$

Now consider the set of axioms $T = \{\tau \mid \mathcal{O}_1 \models \tau \wedge \mathcal{O}_2 \models \tau\}$. Since both $\mathcal{O}_1$ and $\mathcal{O}_2$ entail $\mathcal{O}$ and satisfy every $g \in G$, it is clear that so does T. However,

$$T \not\models \mathcal{O}_1 \tag{25}$$
$$T \not\models \mathcal{O}_2 \tag{26}$$

due to the entailments α (Eq. 23) and β (Eq. 24). Hence T is strictly weaker than both $\mathcal{O}_1$ and $\mathcal{O}_2$. This contradicts the initial assumption of $\mathcal{O}_1, \mathcal{O}_2 \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. ■

Hence applying a GBox G to an ontology $\mathcal{O}$ results in a theory that is unique modulo equivalence, but not necessary finite. As a consequence, we can treat $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ as a single theory when convenient.

Our definition of $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ is strictly semantic, i.e., does not tell us how to identify any $\mathcal{O}' \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. In order to do that, we define a 1-step expansion.

Definition 4.2 The *1-step expansion of $\mathcal{O}$ and G in $\mathcal{L}$*, written $1\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$, is defined as follows:

$$1\mathsf{Exp}(G, \mathcal{O}, \mathcal{L}) = \mathcal{O} \cup \bigcup_{T_B \to T_H \in G} \{T_H\sigma \mid \sigma \in \mathsf{eval}(T_B, \mathcal{O}, \mathcal{L})\}.$$

In other words, we add to $\mathcal{O}$ all instantiated heads of all generators applicable in $\mathcal{O}$. Of course, this extension may result in other generators with other substitutions becoming applicable, and so on recursively.

Lemma 4.3 *If $\mathcal{O}_1 \subseteq \mathcal{O}_2$, then $1\mathsf{Exp}(G, \mathcal{O}_1, \mathcal{L}) \subseteq 1\mathsf{Exp}(G, \mathcal{O}_2, \mathcal{L})$.*

Proof Simple consequence of Def. 3.3 and $\mathsf{eval}(B(g), \mathcal{O}_1, \mathcal{L}) \subseteq \mathsf{eval}(B(g), \mathcal{O}_2, \mathcal{L})$ for any generator g. ■

Definition 4.4 The *n-step expansion of $\mathcal{O}$ and G in $\mathcal{L}$*, written $\mathsf{1Exp}^n(G,\mathcal{O},\mathcal{L})$, is defined as follows:

$$\mathsf{1Exp}^n(G,\mathcal{O},\mathcal{L}) = \underbrace{\mathsf{1Exp}(\ldots\mathsf{1Exp}(}_{n\text{times}}G,\mathcal{O},\mathcal{L})\ldots).$$

We use $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ to denote the least fixpoint of $\mathsf{1Exp}(G,\mathcal{O},\mathcal{L})$. □

Theorem 4.5 *For finite $\mathcal{L}$, the least fixpoint $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ exists and belongs to $\mathsf{Exp}(G,\mathcal{O},\mathcal{L})$.* □

Proof Since $\mathcal{L}$ is finite, the set of all $\mathcal{L}$-substitutions for the variables occurring in G is finite. Let $\Sigma_\mathcal{L}$ be this set, and consider the set $H = \mathcal{O} \cup \bigcup_{T_B \rightarrow T_H \in G, \sigma \in \Sigma_\mathcal{L}} T_H\sigma$, that is, $\mathcal{O}$ as well as all axioms obtained from the heads of instances of generators in G. This set is also finite.

It is easily verified that 1Exp is an operator on the powerset of H. Since 1Exp is monotone, the least fixpoint $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ exists, and belongs to $\mathsf{Exp}(G,\mathcal{O},\mathcal{L})$ by construction.

In other words, our fully semantic definition of $\mathsf{Exp}(G,\mathcal{O},\mathcal{L})$ coincides with the operational semantics based on the fixpoint computation.

Size of the fixpoint For a generator g with variables V, there are at most $|\mathcal{L}|^{|V|}$ different $\mathcal{L}$-substitutions. The size of the fixpoint is therefore bounded by $|G| \times |\mathcal{L}|^n$, where n is the maximum number of variables in any $g \in G$. In the worst case we need to perform entailment checks for all of them, adding one instantiation at a time to $\mathcal{O}$. Hence determining $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ involves up to $(|G| \times |\mathcal{L}|^n)^2$ entailment checks. For finite $\mathcal{L}$ and provided we have a fixed upper bound for n, determining $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ involves a polynomial number of entailment tests and results in a $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ whose size is polynomial in the size of G and $\mathcal{L}$.

Finite vs infinite L

The next examples illustrate the difficulties an infinite language $\mathcal{L}$ can cause. The first example shows how an infinite $\mathcal{L}$ can lead to infinite expansions.

Example 4.6 Consider the ontology $\mathcal{O} = \{\mathsf{A} \sqsubseteq \exists\mathsf{R}.\mathsf{B}\}$, the generator $g : \{?X \sqsubseteq \exists\mathsf{R}.?Y\} \rightarrow \{?X \sqsubseteq \exists\mathsf{R}.\exists\mathsf{R}.?Y\}$, and $\mathcal{L}$ the set of all $\mathcal{EL}$-concept expressions. Clearly, $\mathsf{1Exp}^*(G,\mathcal{O},\mathcal{L})$ is infinite, and so is each expansion in $\mathsf{Exp}(G,\mathcal{O},\mathcal{L})$. □

The next example shows that this does not necessarily happen.

Example 4.7 Consider the ontology $\mathcal{O} = \{\exists \mathsf{R.A} \sqsubseteq \mathsf{A}\}$, the generator $g : \{\exists \mathsf{R}.?X \sqsubseteq ?X\} \rightarrow \{\exists \mathsf{R}.\exists \mathsf{R}.?X \sqsubseteq \exists \mathsf{R}.?X\}$, and $\mathcal{L}$ the set of all $\mathcal{EL}$-concept expressions. Clearly, $1\mathsf{Exp}^*(G, \mathcal{O}, \mathcal{L})$ is infinite, but there is a finite (and equivalent) ontology to this fixpoint in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$, namely $\mathcal{O}$ itself. □

While having to explicitly specify $\mathcal{L}$ may seem to be cumbersome, it is not very restrictive. In fact, it is easy to show that, for finite languages, generators can be rewritten to account for concepts, roles, or individuals that are missing from a given language by grounding the generators.

Definition 4.8 Let $g : T_B \rightarrow T_H$ be a generator, and $\mathcal{L}$ a finite language. The *$\mathcal{L}$-grounding of g* is the finite set of generators $\{T_B\sigma \rightarrow T_H\sigma \mid \sigma \text{ an } \mathcal{L}\text{-substitution}\}$. □

Using $\mathcal{L}$-grounding, we can compensate for a smaller language $\mathcal{L}_1 \subsetneq \mathcal{L}_2$ by $\mathcal{L}_2 \setminus \mathcal{L}_1$-grounding generators, thereby proving the following theorem.

Theorem 4.9 *Let $\mathcal{L}_1 \subseteq \mathcal{L}_2$ be finite languages. For every GBox G there exists a Gbox G' such that, for every $\mathcal{O}$, $\mathcal{O}_1$, $\mathcal{O}_2$ we have that $\mathcal{O}_1 \in \mathsf{Exp}(G', \mathcal{O}, \mathcal{L}_1)$ and $\mathcal{O}_2 \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L}_2)$ implies $\mathcal{O}_1 \equiv \mathcal{O}_2$.* □

Proof Take G' to be the union of the $\mathcal{L}_2$-groundings of every generator in G. ■

Of course, grounding all the generators is a very wasteful way of accounting for a less expressive language. A more clever rewriting algorithm should be possible: for example, if we allow binary conjunctions of names in $\mathcal{L}_2$ but not in $\mathcal{L}_1$, we can add copies of each generator where we replace variables $?X$ with $?X_1 \sqcap ?X_2$.

4.1 GBox containment and equivalence

Having defined GBoxes, we now investigate a suitable notion for containment and equivalence of GBoxes.

Definition 4.10 ($\mathcal{L}$-containment) Let G_1 and G_2 be GBoxes, and $\mathcal{L}$ a language. G_1 is *$\mathcal{L}$-contained in G_2* (written $G_1 \preceq_{\mathcal{L}} G_2$) if

$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models \mathsf{Exp}(G_1, \mathcal{O}, \mathcal{L})$$

for every ontology $\mathcal{O}$. □

The following lemma relating the entailment of theories and the entailment of expansions holds as a direct consequence of the monotonicty of description logics.

Lemma 4.11 *Let G be a GBox, T, T' two theories and $\mathcal{L}$ a language. If $T \models T'$ then $\mathsf{Exp}(G, T, \mathcal{L}) \models \mathsf{Exp}(G, T', \mathcal{L})$.* □

Furthermore, the following is a rather straightforward consequence of the definition of the semantics of generators.

Lemma 4.12 *Let T be a theory, G a GBox, $\mathcal{O}$ an ontology, and $\mathcal{L}$ a language. If $T \models \mathcal{O}$ and T satisfies every generator $g \in G$ then $T \models \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.* □

Using Lemmas 4.11 and 4.12, $\mathcal{L}$-containment can be shown to be decidable, and in fact efficiently so, using a standard freeze technique from database theory.

Theorem 4.13 *Let G_1 and G_2 be GBoxes, and $\mathcal{L}$ a language. G_1 is $\mathcal{L}$-contained in G_2 if and only if $\mathsf{Exp}(G_2, T_B, \mathcal{L}) \models T_H$ for every $T_B \rightarrow T_H \in G_1$.* □

Proof The only-if direction follows directly. For the other direction, by Lemma 4.12 we need to show that if $\mathsf{Exp}(G_2, T_B, \mathcal{L}) \models T_H$ for all $T_B \rightarrow T_H \in G_1$ then for any ontology $\mathcal{O}$

$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models g \text{ for all } g \in G_1, \tag{27}$$
$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models \mathcal{O}. \tag{28}$$

By Lemma 4.12, (27) and (28) imply $\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models \mathsf{Exp}(G_1, \mathcal{O}, \mathcal{L})$, which is the definition of G_1 being $\mathcal{L}$-contained in G_2. (28) is an immediate consequence of the definition of the expansion, hence we only need to show (27).

In the following we slightly abuse notation: $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ for a GBox G, ontology $\mathcal{O}$ and language $\mathcal{L}$ shall refer to an ontology as opposed to a set of possible expansions; by Theorem 4.1, they are all logically equivalent.

Let $T_B \rightarrow T_H \in G_1$ be fixed but arbitrary. Furthermore, let $\sigma \in \mathsf{eval}(T_B, \mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}))$.

Then, by the definition of eval,

$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models T_B\sigma. \tag{*}$$

Applying Lemma 4.11 to (*) yields $\mathsf{Exp}(G_2, \mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}), \mathcal{L}) \models \mathsf{Exp}(G_2, T_B\sigma, \mathcal{L})$. But $\mathsf{Exp}(G_2, \mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}), \mathcal{L}) = \mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L})$ (otherwise $\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L})$ would not be an expansion) and hence

$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models \mathsf{Exp}(G_2, T_B\sigma, \mathcal{L}). \tag{29}$$

Thus what remains is to show that

$$\mathsf{Exp}(G_2, T_B\sigma, \mathcal{L}) \models T_H\sigma, \tag{30}$$

since (29) and (30) together yield

$$\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L}) \models T_H\sigma. \tag{**}$$

which together with (*) implies that $\mathsf{Exp}(G_2, \mathcal{O}, \mathcal{L})$ satisfies $T_B \rightarrow T_H$.

Using compositionality of $\mathcal{L}$-substitutions and the iterative fixpoint construction of the expansion, it is straightforward to show that

$$\mathsf{Exp}(G_2, T_B\sigma, \mathcal{L}) \models \mathsf{Exp}(G_2, T_B, \mathcal{L})\sigma. \tag{31}$$

By the assumption of the theorem, $\mathsf{Exp}(G_2, T_B, \mathcal{L}) \models T_H$ which in turn implies that $\mathsf{Exp}(G_2, T_B, \mathcal{L})\sigma \models T_H\sigma$. This together with (31) yields

$$\mathsf{Exp}(G_2, T_B\sigma, \mathcal{L}) \models (\mathsf{Exp}(G_2, T_B, \mathcal{L})\sigma \models T_H\sigma, \tag{32}$$

thus proving (30) and thereby (**), as desired. ∎

It follows that $\mathcal{L}$-containment is decidable for arbitrary $\mathcal{L}$ (even infinite), since we can restrict ourselves to the language of all subexpressions of $B(G_1)$. Furthermore, the complexity is the same as that of computing an expansion of a GBox.

4.2 GBoxes with negation

In this section we introduce negation-as-failure to GBoxes. We extend the definition of the expansions defined in Section 3, define suitable notions of semi-positive GBoxes and semantics for stratified GBoxes, and prove the corresponding uniqueness results.

To do so, a generator is now a rule of the form $T_B^+(V_1)$, **not** $T_B^-(V_2) \rightarrow T_H(V_3)$, for $T_B^+(V_1), T_B^-(V_2), T_H(V_3)$ templates with $V_3 \subseteq V_1 \cup V_2$. For the sake of notational simplicity, we restrict ourselves here to generators with at most one template in the negative body. It is worth noting, however, that all definitions and results in this section are immediately transferable to generators with multiple templates in the negative bodies (multiple templates in the positive body can of course be simply merged into a single template).

The following definition, together with Definition 3.5 of $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$, provides a minimal model semantics for GBoxes with negation:

Definition 4.14 An ontology $\mathcal{O}$ *satisfies a generator* $g : T_B^+(V_1), \textbf{\textit{not}}\, T_B^-(V_2) \rightarrow T_H(V_3)$ *wrt.* $\mathcal{L}$ if, for every $\sigma \in \mathsf{eval}(T_B^+, \mathcal{O}, \mathcal{L}) \setminus \mathsf{eval}(T_B^-, \mathcal{O}, \mathcal{L})$ we have $\mathcal{O} \models T_H\sigma$. □

Unsurprisingly, adding negation results in the loss of uniqueness of the expansion $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ (cf. Theorem 4.1), as illustrated by the following example.

Example 4.15 Let $\mathcal{L} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{s}\}$, $\mathcal{O} = \{\mathsf{A}(\mathsf{s})\}$ and $G = \{\mathsf{A}(?X), \textbf{not}\, \mathsf{B}(?X) \rightarrow \mathsf{C}(?X)\}$. Then $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ contains the two non-equivalent expansions $\{\mathsf{A}(\mathsf{s}), \mathsf{B}(\mathsf{s})\}$ and $\{\mathsf{A}(\mathsf{s}), \mathsf{C}(\mathsf{s})\}$. □

Next, we extend the definition of the 1-step expansion operator from Definition 4.2 to support negation. However, as Example 4.17 will show, a fixpoint does not always correspond to an expansion in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.

Definition 4.16 The *1-step expansion of* $\mathcal{O}$ *and* G *in* $\mathcal{L}$ of a GBox G with negation, written $1\mathsf{Exp}^-(G, \mathcal{O}, \mathcal{L})$, is defined as follows:

$$1\mathsf{Exp}^-(G, \mathcal{O}, \mathcal{L}) = \mathcal{O} \cup \bigcup_{T_B^+, \textbf{not}\, T_B^- \rightarrow T_H \in G} \{T_H\sigma \mid \sigma \in \mathsf{eval}(T_B^+, \mathcal{O}, \mathcal{L}) \setminus \mathsf{eval}(T_B^-, \mathcal{O}, \mathcal{L})\}.$$

Example 4.17 Consider the ontology $\mathcal{O} = \{\mathsf{Single} \sqsubseteq \mathsf{Person}, \mathsf{Spouse} \sqsubseteq \mathsf{Person}, \mathsf{Single} \sqsubseteq \neg\mathsf{Spouse}, \mathsf{Person}(\mathsf{Maggy})\}$ and the following GBox G

$$\begin{aligned} G = \{ \quad & \{\mathsf{Person}(?X)\}, \textbf{not}\{\mathsf{Single}(?X)\} & \rightarrow \{\mathsf{Spouse}(?X)\}, \\ & \{\mathsf{Person}(?X)\}, \textbf{not}\{\mathsf{Spouse}(?X)\} & \rightarrow \{\mathsf{Single}(?X)\}\} \end{aligned}$$

The expansion $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ contains the two non-equivalent ontologies $\mathcal{O} \cup \{\mathsf{Single}(\mathsf{Maggy})\}$ and $\mathcal{O} \cup \{\mathsf{Spouse}(\mathsf{Maggy})\}$. Furthermore, the iterated fixpoint $(1\mathsf{Exp}^-)^*(G, \mathcal{O}, \mathcal{L})$ is $\mathcal{O} \cup \{\mathsf{Single}(\mathsf{Maggy}), \mathsf{Spouse}(\mathsf{Maggy})\}$; this is, however, not an ontology in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ as it is not entailment-minimal. □

A natural question arising is whether we can identify or even characterize GBoxes with negation that have a unique expansion. To this end, we define suitable notions of semi-positive GBoxes and stratified negation. These are based on the notion of multiple templates affecting others, as formalized next.

Definition 4.18 Let $\mathcal{L}$ be a language, $S = \{S_1, \ldots, S_k\}$ a set of templates, $\mathcal{O}$ an ontology, and T a template. We say that S *activates* T *with respect to* $\mathcal{O}$ *and* $\mathcal{L}$ if there exist $\mathcal{L}$-substitutions $\sigma_1, \ldots \sigma_k$ such that $\mathcal{O} \cup \bigcup S_i\sigma_i \models T\sigma$ for some $\mathcal{L}$-substitution σ. For brevity we omit $\mathcal{O}$ and $\mathcal{L}$ if they are clear from the context. □

In contrast to standard Datalog with negation, the entailment of a template in the body of a generator is not solely dependent on a single generator with a corresponding head firing. Instead, multiple generators might need to fire and interact with $\mathcal{O}$ in order to entail a body template. Hence we use the set S of templates in the definition of activation.

Example 4.19 Consider the GBox containing $g_1\colon T_1(?X) \rightarrow \{?X \sqsubseteq \mathsf{A}\}$, $g_2\colon T_2(?Y) \rightarrow \{?Y \sqsubseteq \mathsf{B}\}$ and $g_3\colon$ **not**$\{?Z \sqsubseteq \mathsf{A} \sqcap \mathsf{B}\} \rightarrow T_3(?Z)$. Then $H(g_1)$ and $H(g_2)$ activate $\{?Z \sqsubseteq \mathsf{A} \sqcap \mathsf{B}\}$ with respect to any $\mathcal{O}$ and $\mathcal{L}$, indicating that the firing of g_3 depends on the combined firing of g_1 and g_2. $\square$

Activation can then be used to define a notion of semi-positive GBoxes, which is analogous to semi-positive Datalog programs.

Definition 4.20 (Semi-positive GBoxes) Let G be a GBox with negation, $\mathcal{L}$ a language, and $\mathcal{O}$ an ontology. G is called *semi-positive w.r.t. $\mathcal{O}$ and $\mathcal{L}$* if no negative body template T_B^- of a generator $g \in G$ is activated by $H(G)$. $\square$

As seen in Example 4.15, even semi-positive GBoxes result in multiple non-equivalent expansions. In that example, neither the ontology $\mathcal{O}$ nor any possible firing of G can yield $B(s)$. As such, we wish to restrict the theories in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ to containing only facts derivable from $\mathcal{O}$ and G. To that end, the following definition suitably restricts the entailment of expansions.

Definition 4.21 Let G be a GBox, $\mathcal{O}$ an ontology, and $\mathcal{L}$ a finite language. We say that an expansion $\mathcal{O}' \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ is *justifiable w.r.t.* $(G, \mathcal{O}, \mathcal{L})$ if the following holds: if $\mathcal{O}' \models T\sigma$ for some template T and substitution σ, then $\mathcal{O} \models T\sigma$ or $H(G)$ activates $T\sigma$ with respect to $\mathcal{O}$ and $\mathcal{L}$. We write simply $\mathcal{O}'$ is *justifiable* when G, $\mathcal{O}$, and $\mathcal{L}$ are clear from the context. $\square$

Using this notion, we can show that, indeed,a GBox being semi-positive implies that its semantics is unambiguous when restricted to justifiable expansions.

Theorem 4.22 *Let G be a semi-positive GBox, $\mathcal{O}$ an ontology, and $\mathcal{L}$ a finite language. Then the fixpoint $(1\mathsf{Exp}^-)^*(G, \mathcal{O}, \mathcal{L})$ exists, is the unique fixpoint of $1\mathsf{Exp}^-$, and is contained in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.* $\square$

Proof Since $1\mathsf{Exp}^-(G, \mathcal{O}, \mathcal{L})$ is an inflationary operator and L is finite, there exists an iterative fixpoint $O^* = (1\mathsf{Exp}^-)^*(G, \mathcal{O}, \mathcal{L})$. By construction, $\mathcal{O}^*$ satisfies $\mathcal{O}$ and all generators $g \in G$ and is justifiable w.r.t. $(G, \mathcal{O}, \mathcal{L})$. We simultaneously prove uniqueness and membership in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ by showing that $O' \models O^*$ for an arbitrary justifiable expansion $O' \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. Let $\mathcal{O}_0 = \mathcal{O}$ and $\mathcal{O}_i = 1\mathsf{Exp}^-(G, \mathcal{O}_{i-1}, \mathcal{L})$ for $i \geq 1$, then $\mathcal{O} = \mathcal{O}_0 \subseteq \ldots \subseteq \mathcal{O}_k = \mathcal{O}^*$ for some k. Assume

$\mathcal{O}_1 \models T\sigma$ for some $\mathcal{L}$-substitution σ and $T \in H(G)$. Then either $\mathcal{O} \models T\sigma$ (in which case $\mathcal{O}' \models T\sigma$) or there exists a generator

$$T_B^+, \mathbf{not}\, T_B^- \rightarrow T$$

such that $\sigma \in \mathsf{eval}(T_B^+, \mathcal{O}, \mathcal{L})$ and $\sigma \notin \mathsf{eval}(T_B^-, \mathcal{O}, \mathcal{L})$. Since G is semi-positive, $H(G)$ cannot activate T_B^-, i.e., there exists no set of generators that, together with the ontology $\mathcal{O}$, could fire in a way that would entail $T_B^-\sigma$. Since $\mathcal{O}'$ is entailment-minimal and justifiable, it must be the case that $\mathcal{O}' \not\models T_B^-\sigma$ and hence $\mathcal{O}' \models T\sigma$. Thus, $\mathcal{O}' \models \mathcal{O}_1$.

The same argument can be applied inductively to show that $\mathcal{O}' \models \mathcal{O}_i$ for $i \geq 1$, thus showing $\mathcal{O}' \models \mathcal{O}^*$. Since $\mathcal{O}'$ was chosen arbitrarily, this proves both the uniqueness and membership claims. ■

The following is a direct corollary of the proof of Theorem 4.22.

Corollary 4.23 *Let G be a semi-positive GBox, $\mathcal{O}$ and ontology and $\mathcal{L}$ a finite language. All justifiable ontologies in* $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ *are logically equivalent.* □

For a GBox to be semi-positive is a very strong requirement. Next, we introduce the notion of a stratified GBox: this does not ensure that all expansions are equivalent, but it ensures that we can determine one of its expansions by expanding strata in the right order. Again, we use $H(G)$ to denote the set of templates in heads of generators in G, and $B(G)$ for the set of templates in (positive or negative) bodies of generators in G.

Definition 4.24 (Stratification) Let $\mathcal{L}$ be a language and $\mathcal{O}$ an ontology. A GBox G is *stratifiable w.r.t. $\mathcal{O}$ and $\mathcal{L}$* if there exists a function $v : H(G) \cup B(G) \rightarrow \mathbb{N}$ such that, for every generator $T_B^+, \mathbf{not}\, T_B^- \rightarrow T_H \in G$ the following holds:

1. $v(T_H) \geq v(T_B^+)$,
2. $v(T_H) > v(T_B^-)$,
3. for every $\subseteq$-minimal $S_1 \subseteq H(G)$ that activates T_B^+, $v(T_B^+) \geq \max_{S' \in S_1} v(S')$,
4. for every $\subseteq$-minimal $S_2 \subseteq H(G)$ that activates T_B^-, $v(T_B^-) > \max_{S' \in S_2} v(S')$. □

The first two conditions in the previous definition are analogous to stratified Datalog, which intuitively states that a body literal must be evaluated (strictly, in the case of negative literals) before head literals. The second two conditions tailor the stratification to generators: generators allow for more interaction amongst their components. As opposed to Datalog, multiple heads combined might be needed to entail a body template. Thus, a body template must be defined in a higher stratum than any possible set of templates that could entail it.

Following this definition, a stratification v of a GBox G w.r.t. an ontology $\mathcal{O}$ gives rise to a partition $G_v^1, \dots G_v^k$ of G, where each generator $g : T_B^+, \mathbf{not}\, T_B^- \rightarrow T_H$ is in the stratum $G_v^{v(T_H)}$.

For a GBox G, an ontology $\mathcal{O}$ and a language $\mathcal{L}$, we can define the *precedence graph* $\mathcal{G}_{G,\mathcal{O},\mathcal{L}}$ as follows: nodes are the templates occuring in G and

1. if $T_B^+, \mathbf{not}\, T_B^- \rightarrow T_H$ is in G, then $\mathcal{G}_{G,\mathcal{O},\mathcal{L}}$ contains the positive edge (T_B^+, T_H) and the negative edge (T_B^-, T_H);

2. for a template T that occurs in the positive (resp. negative) body of a generator and any $\subseteq$-minimal set $\{S_1, \dots, S_k\} \subseteq H(G)$ that activates T w.r.t. $\mathcal{O}$ and $\mathcal{L}$, $\mathcal{G}_{G,\mathcal{O},\mathcal{L}}$ contains the positive (resp. negative) edges (S_i, T) for $1 \leq i \leq k$.

We then get the following classification of stratified GBoxes, the proof of which is entirely analogous to the Datalog case.

Proposition 4.25 *Let $\mathcal{L}$ be a language and $\mathcal{O}$ an ontology. A GBox G is stratifiable w.r.t. $\mathcal{O}$ and $\mathcal{L}$ iff its precedence graph $\mathcal{G}_{G,\mathcal{O},\mathcal{L}}$ has no cycle with a negative edge.* □

Given such a stratification, we can thus define a semantics for stratified negation.

Definition 4.26 (Stratified semantics) Let $\mathcal{O}$ be an ontology, $\mathcal{L}$ a language, and G a GBox stratifiable w.r.t. $\mathcal{O}$ and $\mathcal{L}$. For a stratification v of G and the induced partition $G_v^1, \dots, G_v^k$ of G, we define $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L})$ as follows:

1. $\mathcal{O}_v^1 = \mathcal{O}$,

2. $\mathcal{O}_v^j = 1\mathsf{Exp}^*(G^{j-1}, O^{j-1}, \mathcal{L})$ for $1 < j \leq k$,

3. $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L}) = O_v^k$. □

Theorem 4.27 *Let $\mathcal{O}$ be an ontology, $\mathcal{L}$ a finite language, and G be a GBox stratifiable w.r.t. $\mathcal{O}$ and $\mathcal{L}$. Then $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L})$ exists, is independent of the choice of v, and contained in* $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$. □

Proof Let $G_v^1, \ldots, G_v^k$ be the partitioning of G w.r.t. to a stratification v. By Definition 4.24, each G_v^i is a semi-positive GBox. Hence Theorem 4.22 guarantees the existence of $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L})$. By construction $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L})$ satisfies $\mathcal{O}$ and all generators in G. Furthermore, there cannot exist an ontology $\mathcal{O}'$ such that $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L}) \models \mathcal{O}'$ satisfying $\mathcal{O}$ and all generators in G, as this would contradict the entailment-minimality of the $\mathcal{O}_v^i$.

The proof for the independence of the stratification v is entirely analogous to the Datalog case: the strongly connected components of $\mathcal{G}_{G,\mathcal{O},\mathcal{L}}$ provide the most granular stratification, which can then be used to prove the equivalence of all stratifications (cf. [1] for a proof for stratified Datalog). ■

Remark 4.28 It is worth noting that, although the stratified semantics provides a unique model, stratified GBoxes do not necessarily have a unique expansion. For example, the GBox from Example 4.15 is stratifiable yet has multiple distinct expansions. Moreover, just as in Datalog, there exist nonstratified GBoxes that have a unique expansion.

5 Related work

When combining rules with DL ontologies, the focus has thus far primarily been on (1) encoding ontology axioms in rules for efficient query answering and (2) expanding the expressivity of ontologies using rules. In contrast, GBoxes are designed as a tool for ontology specification by describing instantiation dependencies between templates.

Datalog$^\pm$ [5] falls into the first category: it provides a formalism for unifying ontologies and relational structures. Datalog$^\pm$ captures ontology axioms as rules, and these cannot "add" new axioms.

dl-programs [13] and **DL-safe rules** [34] fall into the second category: dl-programs add nonmonotonic reasoning by means of stable model semantics, whereas DL-safe rules allow for axiom-like rules not expressible in standard DL. However, none of these formalisms adds new TBox axioms to the ontology.

Tawny-OWL[2] and the **Ontology Pre-Processing Language**[3] (OPPL) are formalism for manipulating OWL ontologies [27, 12]. While OPPL was designed to capture patterns and regularities in ontologies, Tawny-OWL is a more general programmatic environment for authoring ontologies that includes powerful support for ontology design patterns. It is part of future work to see

[2]https://github.com/phillord/tawny-owl
[3]http://oppl2.sourceforge.net/index.html

whether GBoxes can be faithfully implemented in Tawny-OWL (OPPL lacks the recursion required).

Another question is whether **metamodeling** in DL, in particular the encoding scheme from [18] can be faithfully captured by (an extension of) GBoxes: this would require *replacing* axioms in $\mathcal{O}$ with others which is currently not supported.

Ontology Design Patterns (ODPs) have been proposed to capture best practices for developing ontologies [15, 4], inspired by Software Design Patterns. While some ODPs are easily expressible in GBoxes, it is part of ongoing work to investigate extensions required to capture others.

Reasonable Ontology Templates[4] (OTTR) [38, 14] provide a framework for macros in OWL ontologies, based on the notion of templates. In contrast to GBoxes, "matching" of templates is defined syntacically and non-recursively, but they can be named and composed to give rise to more complex templates.

The **Generalized Distributed Ontology, Modelling and Specification Language** (GDOL) [24] is a formalism facilitating the template-based construction of ontologies from a wide range of logics. In addition to concepts, roles, and individuals, parameters may be ontologies which act as preconditions for template instantiation: for a given substitution, the resulting parameter ontology must be satisfiable in order to instantiate the template. Thus these preconditions serve only as a means to restrict the set of allowed instantiations of a template, whereas in GBoxes, an ontology triggers such substitutions.

6 Future work

We have presented first results about a template-based language for capturing recurring ontology patterns and using these to specify larger ontologies. Here, we list some areas that we would like to investigate in the future.

Finite representability In general, the semantics of GBoxes is such that the expansion of a GBox and ontology can be infinite if the substitution range given by $\mathcal{L}$ is infinite. A natural question arising is whether/which other mechanisms can ensure that *some* expansion is finite, and how can we compute such a finite expansion? Furthermore, given $G, \mathcal{O}, \mathcal{L}$, when can we decide whether an ontology in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ is finite?

Controlling substitutions So far, we have only considered entailment for generators when determining matching substitutions. Consider the ontology

[4] http://ottr.xyz

$\mathcal{O} = \{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \mathsf{C}\}$ and the template $?X \sqsubseteq C$. The resulting substitutions include concepts A and B, but also a multitude of possibly unwanted, redundant concepts, e.g., $\{\mathsf{A} \sqcap \mathsf{A}, \mathsf{A} \sqcap \mathsf{B}, \ldots\}$. Hence restricting substitutions to "reasonable" or possible "parametrizable" (e.g., maximally general) ones is part of future work.

Entailment problems for ontologies with Gboxes The expansion of a Gbox over an ontology is itself an ontology and can be used as such for standard reasoning tasks. A question of interest is whether/how reasoning on the input ontology and GBox directly, without computing an expansion, can improve reasoning efficiency.

Furthermore, there are plenty of reasoning tasks about GBoxes which naturally reduce to reasoning tasks over ontologies. For example, checking whether a single generator $g : T_B \to T_H$ always leads to inconsistency is equivalent to checking whether $T_B \cup T_H$ is inconsistent. This generalizes to similar questions over entire GBoxes: To check whether there exists an ontology $\mathcal{O}$ such that every generator g in a GBox G fires, it suffices to check that the union of the generators' bodies is consistent.

However, there are also global properties of Gboxes that do not reduce to individual templates. For example, do two GBoxes G_1 and G_2 specify equivalent ontologies? While Section 4.1 contains some results about such problems, we believe there is more to do here.

Extensions to generators Another area of future work is motivated by our preliminary analysis of *logical* ontology design patterns [17]. We found that a number of rather straightforward, seemingly useful such pattern require some form of ellipses and/or maximality. Consider, for example, the role closure pattern on the role $\mathsf{hasTopping}$: if $\mathcal{O}$ entails that $\mathsf{MyPizza} \sqsubseteq \exists \mathsf{hasTopping}.X_1 \sqcap \ldots \exists \mathsf{hasTopping}.X_n$ and n is maximal for pairwise incomparable X_i, then we would like to automatically add $\mathsf{MyPizza} \sqsubseteq \forall \mathsf{hasTopping}.(X_1 \sqcup \ldots \sqcup X_n)$. Extending generators to capture some form of ellipses or unknown number of variables and maximality conditions on substitutions for variables will be part of future work.

For GBoxes to be indeed intention revealing, we will also support named generators and named sets of axioms in the body or the head of generators, as in OTTR [38].

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Eva Blomqvist. Fully automatic construction of enterprise ontologies using design patterns: Initial method and first experiences. In *OTM Conferences (2)*, volume 3761 of *Lecture Notes in Computer Science*, pages 1314–1329. Springer, 2005.

[4] Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In *ICEIS (3)*, pages 413–416, 2005.

[5] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. Datalog+/- : A family of languages for ontology querying. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Datalog Reloaded - 1st International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, volume 6702 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2011.

[6] Peter Clark. Knowledge patterns. In *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2008.

[7] Ricardo de Almeida Falbo, Monalessa Perini Barcellos, Julio Cesar Nardi, and Giancarlo Guizzardi. Organizing ontology design patterns as ontology pattern languages. In *ESWC*, volume 7882 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2013.

[8] Ricardo de Almeida Falbo, Monalessa Perini Barcellos, Fabiano Borges Ruy, Giancarlo Guizzardi, and Renata S. S. Guizzardi. Ontology pattern languages. In *Ontology Engineering with Ontology Design Patterns*, volume 25 of *Studies on the Semantic Web*, pages 133–159. IOS Press, 2016.

[9] Ricardo de Almeida Falbo, Glaice Kelly Quirino, Julio Cesar Nardi, Monalessa Perini Barcellos, Giancarlo Guizzardi, Nicola Guarino, Antonella Longo, and Barbara Livieri. An ontology pattern language for service modeling. In *SAC*, pages 321–326. ACM, 2016.

[10] Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Higher-order description logics for domain metamodeling. In *AAAI*. AAAI Press, 2011.

[11] Érica Ferreira de Souza, Ricardo de Almeida Falbo, and Nandamudi L. Vijaykumar. Using ontology patterns for building a reference software testing ontology. In *EDOC Workshops*, pages 21–30. IEEE Computer Society, 2013.

[12] Mikel Egaña, Robert Stevens, and Erick Antezana. Transforming the axiomisation of ontologies: The ontology pre-processor language. In *OWLED (Spring)*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[13] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12):1495 – 1539, 2008.

[14] Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen. Reasonable Macros for Ontology Construction and Maintenance. In *DL Workshop*, 2017.

[15] Aldo Gangemi. Ontology design patterns for semantic web content. In *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2005.

[16] Aldo Gangemi, Nicola Guarino, Claudio Masolo, and Alessandro Oltramari. Understanding top-level ontological distinctions. In *OIS@IJCAI*, volume 47 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[17] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.

[18] Birte Glimm, Sebastian Rudolph, and Johanna Völker. Integrated metamodeling and diagnosis in OWL 2. In *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2010.

[19] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *J. Web Sem.*, 6(4):309–322, 2008.

[20] Giancarlo Guizzardi. Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In *ER*, volume 8824 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2014.

[21] Karl Hammar, Eva Blomqvist, David Carral, Marieke van Erp, Antske Fokkens, Aldo Gangemi, Willem Robert van Hage, Pascal Hitzler, Krzysztof Janowicz, Nazifa Karima, Adila Krisnadhi, Tom Narock, Roxane Segers, Monika Solanki, and Vojtech Svátek. Collected research questions concerning ontology design patterns. In *Ontology Engineering with Ontology Design Patterns*, volume 25 of *Studies on the Semantic Web*, pages 189–198. IOS Press, 2016.

[22] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors. *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press, 2016.

[23] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Alfa Krisnadhi, and Valentina Presutti. Towards a simple but useful ontology design pattern representation language. In *WOP@ISWC*, volume 2043 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[24] Bernd Krieg-Brückner and Till Mossakowski. Generic ontologies and generic ontology design patterns. In *WOP@ISWC*, 2017.

[25] Adila Krisnadhi, Yingjie Hu, Krzysztof Janowicz, Pascal Hitzler, Robert A. Arko, Suzanne Carbotte, Cynthia Chandler, Michelle Cheatham, Douglas Fils, Timothy W. Finin, Peng Ji, Matthew B. Jones, Nazifa Karima, Kerstin A. Lehnert, Audrey Mickle, Thomas W. Narock, Margaret O'Brien, Lisa Raymond, Adam Shepherd, Mark Schildhauer, and Peter Wiebe. The geolink modular oceanography ontology. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 2015.

[26] Petra Kubincová, Jan Kluka, and Martin Homola. Expressive description logic with instantiation metamodelling. In *KR*, pages 569–572. AAAI Press, 2016.

[27] Phillip Lord. The semantic web takes wing: Programming ontologies with tawny-owl. In *OWLED*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[28] Catalina Martínez-Costa, Daniel Karlsson, and Stefan Schulz. Ontology patterns for clinical information modelling. In *WOP*, volume 1302 of *CEUR Workshop Proceedings*, pages 61–72. CEUR-WS.org, 2014.

[29] Eleni Mikroyannidi, Luigi Iannone, Robert Stevens, and Alan L. Rector. Inspecting regularities in ontology design using clustering. In *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 438–453. Springer, 2011.

[30] Eleni Mikroyannidi, Nor Azlinayati Abdul Manaf, Luigi Iannone, and Robert Stevens. Analysing syntactic regularities in ontologies. In *OWLED*, volume 849 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[31] Eleni Mikroyannidi, Manuel Quesada-Martínez, Dmitry Tsarkov, Jesualdo Tomás Fernández-Breis, Robert Stevens, and Ignazio Palmisano. A quality assurance workflow for ontologies based on semantic regularities. In *EKAW*, volume 8876 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2014.

[32] Till Mossakowski. The distributed ontology, model and specification language – DOL. In Phillip James and Markus Roggenbach, editors, *Recent Trends in Algebraic Development Techniques*, pages 5–10, Cham, 2017. Springer International Publishing.

[33] Boris Motik. On the properties of metamodeling in OWL. *J. Log. Comput.*, 17(4):617–637, 2007.

[34] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41 – 60, 2005. Rules Systems.

[35] Regina Motz. OWL extended with meta-modelling. In *ISW-LOD@IBERAMIA*, volume 1807 of *CEUR Workshop Proceedings*, pages 55–60. CEUR-WS.org, 2016.

[36] Martin O'Connor, Holger Knublauch, Samson Tu, Benjamin Grosof, Mike Dean, William Grosso, and Mark Musen. Supporting rule system interoperability on the semantic web with swrl. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web – ISWC 2005*, pages 974–986, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[37] Glaice K. S. Quirino, Monalessa Perini Barcellos, and Ricardo de Almeida Falbo. OPL-ML: A modeling language for representing ontology pattern languages. In *ER Workshops*, volume 10651 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2017.

[38] Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. Practical ontology pattern instantiation, discovery, and maintanence with reasonable ontology templates. Accepted for ISWC 2018 research track, 2018.

[39] Steffen Staab, Michael Erdmann, and Alexander Maedche. Engineering ontologies using semantic patterns. In *OIS@IJCAI*, volume 47 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[40] Ondrej Sváb-Zamazal, Vojtech Svátek, and Luigi Iannone. Pattern-based ontology transformation service exploiting OPPL and OWL-API. In *EKAW*, volume 6317 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2010.

[41] Eduardo Zambon and Giancarlo Guizzardi. Formal definition of a general ontology pattern language using a graph grammar. In *FedCSIS*, pages 1–10, 2017.