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

Christian Kindermann[1], Daniel P. Lupp[2], Uli Sattler[1], and Evgenij Thorstensen[2]

[1] University of Manchester, UK
[2] University of Oslo, Norway

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 [4], 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 [2, 9, 13]. Another view, emphasizing common ontological distinctions, led to the emergence of *Upper Ontologies* which aim to categorize general ideas shareable across different domains [10]. Orthogonal to such conceptual patterns, the existence of syntactic regularities in ontologies has been noted and some aspects of their nature have been analyzed [16–18].

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, which we show 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. The goal is 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

The second author was supported by Norwegian Research Council grant no. 230525. The fourth author partially funded by the Norwegian Research Council through the SIRIUS Centre for Research-based Innovation (grant no. 237898).

is not possible to group conceptually related axioms or indicate interdependencies between axioms. While ontology editors such as Protégé[3] display an ontology through a hierarchy of its entities, conceptual interdependencies between axioms are hidden and the underlying structural design of an ontology remains obscured.

Example 1. Consider the ontology

$$\mathcal{O}_1 = \{\mathsf{Jaguar} \sqsubseteq \mathsf{Animal}, \quad \mathsf{Jaguar} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Jaguar}, \tag{1}$$
$$\mathsf{Tiger} \sqsubseteq \mathsf{Animal}, \quad \mathsf{Tiger} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Tiger}, \tag{2}$$
$$\mathsf{Lion} \sqsubseteq \mathsf{Animal}, \quad \mathsf{Lion} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Lion}\} \tag{3}$$

Then, an ontology editor will group the entities $\mathsf{Jaguar}, \mathsf{Tiger}$ 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 explicitly has a potential to reveal some aspects of the intentions for the design of an ontology.

Example 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 from the signature of $\mathcal{O}_2$ 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.

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.

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

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.

This paper is accompanied by a more detailed technical report [7] providing more elaborate examples, a discussion of potential use cases, a list of open questions, and all omitted proofs.

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 [1]. 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. As the variables are concept, role, and individual names, the semantics of a template is the same as of a regular ontology. 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 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. $g\colon \{?X \sqsubseteq \mathsf{Animal}\} \to \{?X \sqsubseteq \forall \mathsf{hasChild}.?X\}$ 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 [20], as OTTR template semantics is defined syntactically.

Definition 2. *Let $g\colon T_B(V_B) \to T_H(V_H)$ 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 $\mathcal{O} \models T_H\sigma$.*

Example 4. Consider the generator g from Example 3, and let $\mathcal{L}$ be the language of all concept names. 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 wrt. $\mathcal{L}$, while the theory $\mathcal{O}_2 = \{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}\}$ does not.

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. *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).

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 5. Recall the generator g from Example 3, 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 $\{\mathsf{Turtle} \sqsubseteq \mathsf{Mammal}, \mathsf{Mammal} \sqsubseteq \mathsf{Animal}, \mathsf{Turtle} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Turtle}, \mathsf{Mammal} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Mammal}\} \in \mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.

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 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.*

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

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

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

Definition 5. *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:*

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

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

Theorem 2. *For finite* $\mathcal{L}$, *the least fixpoint* $\mathsf{1Exp}^*(G, \mathcal{O}, \mathcal{L})$ *exists and belongs to* $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.

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 6. Consider the ontology $\mathcal{O} = \{\mathsf{A} \sqsubseteq \exists \mathsf{R}.\mathsf{B}\}$, the generator $g : \{?X \sqsubseteq \exists \mathsf{R}.?Y\} \to \{?X \sqsubseteq \exists \mathsf{R}.\exists \mathsf{R}.?Y\}$, 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 7. Consider the ontology $\mathcal{O} = \{\exists \mathsf{R}.\mathsf{A} \sqsubseteq \mathsf{A}\}$, the generator $g : \{\exists \mathsf{R}.?X \sqsubseteq ?X\} \to \{\exists \mathsf{R}.\exists \mathsf{R}.?X \sqsubseteq \exists \mathsf{R}.?X\}$, and $\mathcal{L}$ the set of all $\mathcal{EL}$-concept expressions. Clearly, $\mathsf{1Exp}^*(G, \mathcal{O}, \mathcal{L})$ 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 6. *Let $g : T_B \to T_H$ be a generator, and $\mathcal{L}$ a finite language. The $\mathcal{L}$-*grounding of *g is the finite set of generators $\{T_B\sigma \to T_H\sigma \mid \sigma$ an $\mathcal{L}$-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 3. *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 $\mathcal{O}_1 \equiv \mathcal{O}_2$.*

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 define a suitable notion for containment and equivalence of GBoxes.

Definition 7 ($\mathcal{L}$-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 2. *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 3. *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 2 and 3, $\mathcal{L}$-containment can be shown to be decidable, and in fact efficiently so, using a standard freeze technique from database theory.

Theorem 4. *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 $T_B \to T_H \in G_1$.*

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 $T_B^+(V_1), \mathbf{not}\, T_B^-(V_2) \to T_H(V_3)$, 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 of $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$, provides a minimal model semantics for GBoxes with negation:

Definition 8. *A theory* $\mathcal{O}$ satisfies a generator $g : T_B^+(V_1), \mathbf{not}\, T_B^-(V_2) \to T_H(V_3)$ 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 1), as illustrated by the following example.

Example 8. Let $\mathcal{L} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{s}\}$, $\mathcal{O} = \{\mathsf{A(s)}\}$ and $G = \{\mathsf{A}(?X), \mathbf{not}\, \mathsf{B}(?X) \to \mathsf{C}(?X)\}$. Then $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ contains the two non-equivalent expansions $\{\mathsf{A(s)}, \mathsf{B(s)}\}$ and $\{\mathsf{A(s)}, \mathsf{C(s)}\}$.

Next, we extend the definition of the 1-step expansion operator from Definition 4 to support negation. However, as Example 9 will show, a fixpoint does not always correspond to an expansion in $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.

Definition 9. *The* 1-step expansion of $\mathcal{O}$ and G in $\mathcal{L}$ *of a GBox* G *with negation, written* $\mathsf{1Exp}^-(G, \mathcal{O}, \mathcal{L})$*, is defined as follows:*

$$\mathsf{1Exp}^-(G, \mathcal{O}, \mathcal{L}) = \mathcal{O} \cup \bigcup_{T_B^+, \mathbf{not}\, T_B^- \to T_H \in G} \{T_H\sigma \mid \sigma \in \mathsf{eval}(T_B^+, \mathcal{O}, \mathcal{L}) \setminus \mathsf{eval}(T_B^-, \mathcal{O}, \mathcal{L})\}.$$

Example 9. Consider the ontology $\mathcal{O} = \{\mathsf{Single} \sqsubseteq \mathsf{Person}, \mathsf{Spouse} \sqsubseteq \mathsf{Person}, \mathsf{Single} \sqsubseteq \neg\mathsf{Spouse}, \mathsf{Person(Maggy)}\}$ and the following GBox G

$$G = \{\ \{\mathsf{Person}(?X)\}, \mathbf{not}\{\mathsf{Single}(?X)\} \to \{\mathsf{Spouse}(?X)\},\\ \{\mathsf{Person}(?X)\}, \mathbf{not}\{\mathsf{Spouse}(?X)\} \to \{\mathsf{Single}(?X)\}\}$$

The expansion $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$ contains the two non-equivalent ontologies $\mathcal{O} \cup \{\mathsf{Single(Maggy)}\}$ and $\mathcal{O} \cup \{\mathsf{Spouse(Maggy)}\}$. Furthermore, the iterated fixpoint $(\mathsf{1Exp}^-)^*(G, \mathcal{O}, \mathcal{L})$ is $\mathcal{O} \cup \{\mathsf{Single(Maggy)}, \mathsf{Spouse(Maggy)}\}$; 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 10. *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 10. Consider the GBox containing generators $g_1 \colon T_1(?X) \to \{?X \sqsubseteq \mathsf{A}\}, g_2 \colon T_2(?Y) \to \{?Y \sqsubseteq \mathsf{B}\}$ and $g_3 \colon \mathbf{not}\{?Z \sqsubseteq \mathsf{A} \sqcap \mathsf{B}\} \to T_3(?Z)$. 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.

Activation can then be used to define a notion of semi-positive GBoxes, which is analogous to semi-positive Datalog programs.

Definition 11 (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 $H(G)$.*

As seen in example 8, 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 12. *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.*

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 5. *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 $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.*

The following is a direct corollary of the proof of Theorem 5.

Corollary 1. *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 13 (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 $v : H(G) \cup B(G) \to \mathbb{N}$ such that, for every generator $T_B^+, \textbf{not}\, T_B^- \to T_H \in G$ 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 $g : T_B^+, \textbf{not}\, T_B^- \to T_H$ 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 $T_B^+, \textbf{not}\, T_B^- \to T_H$ 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 1. *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 14 (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. $\mathcal{O}_v^j = \mathsf{1Exp}^*(G^{j-1}, O^{j-1}, \mathcal{L})$ *for* $1 < j \leq k$,
3. $\mathcal{O}_v^{\mathsf{strat}}(G, \mathcal{O}, \mathcal{L}) = O_v^k$.

Theorem 6. *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 $\mathsf{Exp}(G, \mathcal{O}, \mathcal{L})$.*

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$^{\pm}$ [3] 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 [6] and **DL-safe rules** [19] 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[4] and the **Ontology Pre-Processing Language**[5] (OPPL) are formalism for manipulating OWL ontologies [5,15]. 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 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 [12] 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 [2,9], 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[6] (OTTR) [8,20] 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) [14] 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.

[4] https://github.com/phillord/tawny-owl
[5] http://oppl2.sourceforge.net/index.html
[6] http://ottr.xyz

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 $\mathcal{O} = \{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \mathsf{C}\}$ 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 $g\colon T_B \to T_H$ 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 [11]. We found that a number of rather straightforward, seemingly useful such patterns 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 [20].

References

1. 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.
2. Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In *ICEIS (3)*, pages 413–416, 2005.
3. 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.
4. Peter Clark. Knowledge patterns. In *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2008.
5. 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.
6. 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.
7. Henrik Forssell, Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen. Generating ontologies from templates: A rule-based approach for capturing regularity. Technical report, 2018. https://arxiv.org/abs/1809.10436v1.
8. Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen. Reasonable Macros for Ontology Construction and Maintenance. In *DL Workshop*, 2017.
9. 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.
10. 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.
11. Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.
12. 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.
13. 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.
14. Bernd Krieg-Brückner and Till Mossakowski. Generic ontologies and generic ontology design patterns. In *WOP@ISWC*, 2017.
15. Phillip Lord. The semantic web takes wing: Programming ontologies with tawny-owl. In *OWLED*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
16. 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.
17. 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.

18. 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.
19. 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.
20. 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.