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 $\mathcal{EL}_\bot$ [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.	$\exists R.C$	$\{x \mid \exists y \langle x, y\rangle \in R^I \wedge y \in C^I\}$	$\exists(R\sigma).(C\sigma)$
Univ. restr.	$\forall R.C$	$\{x \mid \forall y \langle x, y\rangle \in R^I \rightarrow y \in C^I\}$	$\forall(R\sigma).(C\sigma)$
Card. restr.	$\geq nR.C$	$\{x \mid \#\{y \mid \langle x, y\rangle \in R^I \wedge y \in C^I\} \geq n\}$	$\geq n(R\sigma).(C\sigma)$
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 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.

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

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 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 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 PartLength(whole, part, length) is

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

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 $\mathsf{eval}(T, O) = \{\sigma|_{\mathsf{param}(T)} \mid T\sigma \text{ a satisfiable instance such that } T\sigma \subseteq O\}$. 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 $T_1\sigma' \not\subseteq T_2\sigma'$.

$2 \rightarrow 3$: 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 $T_1\sigma' \subseteq T_2\sigma'$.

$3 \rightarrow 1$: 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 $O \models T\sigma$.

Example 6. Let T be the template $\mathsf{P(x,y)} \mapsto \{\mathsf{x} \sqsubseteq \mathsf{y}\}$ 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 $\mathsf{eval}_{\models}(T, O) = \{\{x \mapsto \phi, y \mapsto \psi\} \mid \phi, \psi\ L\text{-concept expressions, and } O \models \phi \sqsubseteq \psi\}$. 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 co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015.*, 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, published online: http://ontologydesignpatterns.org/wiki/images/1/11/WOP2016_paper_01.pdf.
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.