Mapping Natural Language to Description Logic

Bikash Gyawali, Anastasia Shimorina, Claire Gardent, Samuel Cruz-Lara, and Mariem Mahfoudh

CNRS/LORIA, Nancy, France
firstname.lastname@loria.fr

Abstract. While much work on automated ontology enrichment has focused on mining text for concepts and relations, little attention has been paid to the task of enriching ontologies with complex axioms. In this paper, we focus on a form of text that is frequent in industry, namely system installation design principle (SIDP) and we present a framework which can be used both to map SIDPs to OWL DL axioms and to assess the quality of these automatically derived axioms. We present experimental results on a set of 960 SIDPs provided by Airbus which demonstrate (i) that the approach is robust (97.50% of the SIDPs can be parsed) and (ii) that DL axioms assigned to full parses are very likely to be correct in 96% of the cases.

Keywords: Natural Language Processing, OWL, Quality Checks

1 Introduction

While there has been much work on enriching ontologies from texts [14, 23], most of this work focuses on concepts and relations. As noted in [21], "ontology learning from text is mostly restricted to inexpressive ontologies while the acquisition of complex axioms involving logical connectives, role restrictions and other expressive features of OWL remains largely unexplored."

There are several reasons why addressing this bottleneck is important.

First, manually creating ontologies is a difficult and time consuming task which requires a high level of domain knowledge and technical expertise. Being able to automate or semi-automate the enrichment of ontologies with complex axioms would help diminish the time and expertise required for building the knowledge bases required by semantic applications.

Second, being able to enrich an ontology using complex axioms derived from text would permit semantic reasoning on the content of that text. In particular, it would permit querying the content of that text in the context of the background knowledge encoded in the ontology (using e.g., conjunctive tree queries on the enriched ontology). It would also allow for consistency checking. Is the text consistent with the knowledge contained in the initial ontology? Is a set of text consistent both internally (are all texts consistent with each other?) and externally (is each text in this set consistent with the ontology?).

Third, it would help bridge the gap between a document-centric and a model-centric view of information. In industries and in governmental services and European organizations, a great number of technical documents (i.e. documents meant to pass information in a way as less ambiguous as possible) are manipulated. Alternatively to this text-based document approach, there is also a use of multiple kinds of "models" (Description Logic Knowledge Bases, UML diagrams, Enterprise Architecture diagrams, etc.), which have increasing popularity. Document-centric and model-centric approaches are, nowadays, largely disconnected however. Being able to automatically map a text to the corresponding model would facilitate the task of technical authors. This would mean that not only the human-friendly text is distributed, but also the associated computer-processable equivalent (model) of the text content.

In this paper, we consider the task of enriching an existing OWL DL (Description Logic) ontology with complex axioms derived from text. We focus on a form of text that is frequent in industry, namely system installation design principles (SIDP) such as (1a-b).

(1) a. Pipes shall be identified with labels.
b. Spacer shall be used only with attachment device to increase distance with regards to structure, bundles or other systems.

System installation design principles are regulations and directives about how to install a system or a set of systems in a functional area (e.g., electrical and optical system or Water Waste System). For instance, at Airbus, for each aircraft project, a set of SIDPs is produced to ensure that planes comply with all system requirements and take into consideration applicable regulations and procedures. In what follows, we distinguish between *simple SIDPs* consisting of a single clause (e.g., 1a) and *complex SIDPs* which involve more than one clause (e.g., 1b).

Our contribution is threefold.

- **Semantic Parsing.** We propose a framework for mapping SIDPs to OWL DL axioms which combines an automatically derived lexicon, a small hand-written grammar, a parser and a surface realiser. This framework is modular, robust and reversible. It is modular in that, different lexicons or grammars may be plugged to meet the requirements of the semantic application being considered. For instance, the lexicon (which relates words and concepts) could be built using a concept extraction tool, i.e. a text mining tool that extracts concepts from text (e.g., [2]). And the grammar could be replaced by a grammar describing the syntax of other document styles such as cooking recipes. It is robust in that, in the presence of unknown words, the parser can skip words and deliver a connected (partial) parse. And, it is reversible in that the same grammar and lexicon can be used both for parsing and for generation.
- **Quality Assessment.** We provide a method for assessing the system output. A chief difficulty when mapping text to semantic representations is the

lack of accepted criteria for assessing the correctness of the semantic representations derived by the system. We tackle this issue by exploiting techniques from both natural language generation and automated reasoning. We evaluate our approach on a dataset of 960 System Installation Design Principles and report results on coverage (proportion of SIDP parsed) and quality assessment (BLEU score for re-generated sentences, statistics on syntactic well-formedness). In particular, we show that the system provides a DL formula for 97.50% of the input SIDPs and that the DL formulae assigned to full parses are very likely to be correct in 96% of the cases.

– **Ontology Enrichment.** We show that the output of our semantic parser can be used to enrich an existing ontology.

2 Related Work

[21] propose a method for converting natural language definitions to complex DL axioms by first, parsing definitions and second, applying ad hoc transformation rules to parse trees. One main difference with our approach is that we use a generic framework for semantic parsing instead. This allows for a more modular and principled approach (for instance, different semantics could be experimented with). This permits the exploitation of well-understood, highly optimised parsing algorithms (we use a standard CKY algorithms extended to increase robustness to unknown words). And this supports a mapping between natural language and logical formulae that is both direct and reversible – by contrast, [21]'s two step approach (dependency parsing followed by the application of transformation rules) makes it more difficult to identify sources of errors as re-generation cannot be used to "visualise" the natural language content of the derived semantic formulae.

More recently, [17] proposed a neural semantic parser which derives DL formulae from natural language definitions. One important difference with our approach is that the data they are training on has been synthesised. In particular, the input text has been authored using ACE (Attempto Controlled English [13]) and grammar based generation. In contrast, we work on human authored text.

Our work also differs from approaches such as [20, 1] which support the authoring of complex DL axioms. In these approaches, DL axioms are authored in an interactive fashion using controlled languages. The main difference with our work is that our approach does not require the use of a control language. Instead, we provide a framework for automatically parsing uncontrolled natural language as can be found for instance, in industrial documentation.

Finally, there has been much work on extracting knowledge from dictionaries [3, 18] but as mentioned in the introduction, in these approaches, the knowledge derived is restricted to the lower layer of the "ontology-learning layer cake" [3] and thus focuses mostly on extracting concepts and relations rather than complex axioms.

3 Approach Overview

A central bottleneck when developing semantic parsers and text generators is the lack of parallel corpus aligning text and semantic representations. Such corpora are not generally available and are costly and difficult to build (humans find it difficult to associate a text with a logical formula representing its meaning). Moreover, as noted in [22], even if such corpora can be built, their logical coverage is often restricted and so any semantic parser trained on them will fail to analyse logical structures missing from the training data.

We therefore explore an alternative approach in which we combine a symbolic grammar-based approach with an automatically acquired lexicon and a robust parsing algorithm which can skip unknown words to produce connected, possibly partial, parses. The approach also integrates a surface realiser which given a DL formula can produce a text expressing the meaning of that formula. Figure 1 outlines our approach showing the interaction of various components. The lexicon maps verbs and noun phrases to complex and simple concepts respectively. The grammar provides a declarative specification of how text relates to meaning (as represented by OWL DL formulae). The parser and the generator exploit the grammar and the lexicon to map natural language to OWL DL formulae (semantic parsing) and OWL DL formulae to natural language (generation and more specifically, surface realisation), respectively.

In the following sections, we describe each of these components in detail. Section 4 summarises the results obtained when processing 960 SIDPs from Airbus to enrich an existing ontology developed by Airbus engineers.

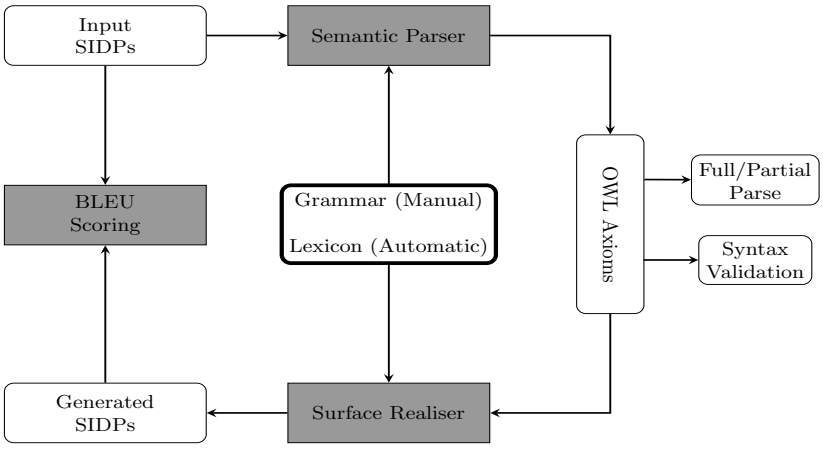


Fig. 1. Parsing and Generation of Airbus SIDPs.

3.1 Grammar

The grammar provides a declarative specification of the relation between natural language phrases and Description Logic formulae. We use a Feature-Based Lexicalised Tree Adjoining Grammar augmented with a unification-based flat semantics (FB-LTAG, [7]). We start by introducing FB-LTAG. We then define the semantic representation language it integrates and its mapping to OWL. Finally, we show how lexical and grammatical knowledge can be dissociated thereby allowing for increased genericity.

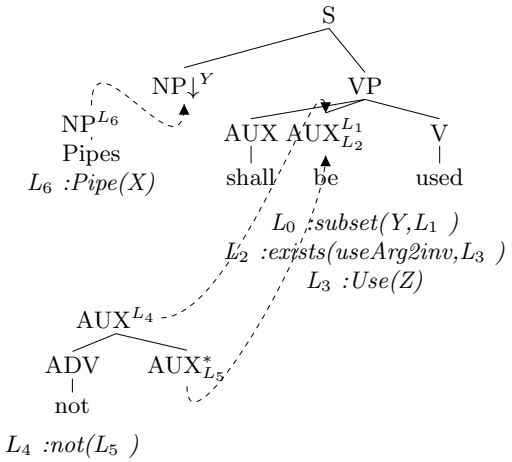


Fig. 2. Example FB-LTAG with Unification-Based Semantics. The variables decorating the tree nodes (e.g., X) abbreviate feature structures of the form $[idx : X]$ where X is a unification variable.

FB-LTAG. Figure 2 shows an illustrating FB-LTAG. In essence[1], an FB-LTAG with unification semantics consists of a set of (trees, semantics) pairs where tree nodes may be labelled with non recursive feature structures (sets of feature-value pairs where values can be constants or unification variables) and semantics may contain unification variables shared with variables occurring in the corresponding tree. During parsing, trees are combined using the grammar operations (adjunction and substitution) and unification is applied to both the feature structures in the tree and the literals in the semantic representation. The semantics of a derived tree is the union of the semantics of the trees contributing to its derivation modulo unification. For instance, given the sentence *Pipes shall not be used*, the combination of the three trees shown in Figure 2 will yield the derived tree and the semantics shown in Figure 3. That is, the grammar assigns to sentence

[1] See [7] for a more precise definition of the FB-LTAG framework.

(2), a flat semantic formula which is equivalent to the DL formula (2a) whose interpretation (2b)[2] can be glossed as (2c) or more simply, (2d).

(2) Pipes shall not be used
a. $Pipe \sqsubseteq \neg\exists useArg2^{-}.(Use)$
b. $\{x \mid x \in Pipe\} \subseteq D^I \setminus \{y \mid (x,y) \in useArg2 \wedge x \in Use\}$
c. *Pipes are not in the set of things that are the arg2 participant of a Use event*
d. *Pipes are not things that are used*

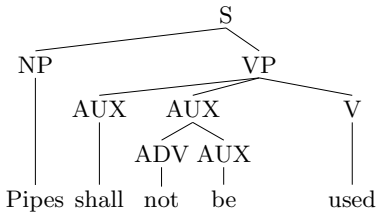
Semantic Representation Language. In the grammar, the semantic representation language used is a flattened version of description logic where subformulae are associated with labels and labels substituted for subformulae. For instance, the flattened version of the DL formula *C1* $\sqsubseteq \exists$ *R.C2* is l_0*:subset(*l_1*,*l_2*),* l_1*:C1,* l_2*:exists(*l_3*),* l_3*:C2*. We convert the flat semantic representations output by the parser to OWL functional syntax using the mapping shown below where $\tau(X)$ is the DL conversion of X, l_i are labels, C_i are arbitrarily complex DL concepts, and R are DL roles.

$$\tau(\phi) = \begin{cases} \text{ObjectSomeValuesFrom(:R } \tau(C)) & \text{if } \phi = l_i : exists(R, l_j) \;\; l_j : C \\ \text{SubClassOf}(\tau(C_1)\ \tau(C_2)) & \text{if } \phi = l_i : subset(l_j, l_k) \;\; l_j : C_1 \;\; l_k : C_2 \\ \text{ObjectIntersectionOf}(\tau(C_1)\ \tau(C_2)) & \text{if } \phi = l_i : and(l_j, l_k) \;\; l_j : C_1 \;\; l_k : C_2 \\ (\tau(C1) \sqcap \tau(C2)) & \text{if } \phi = l_i : and(l_j, l_k) \;\; l_j : C1 \;\; l_k : C2 \\ (\tau(C1) \sqcup \tau(C2)) & \text{if } \phi = l_i : or(l_j, l_k) \;\; l_j : C1 \;\; l_k : C2 \\ \text{not}(\tau(C)) & \text{if } \phi = l_i : not(l_j) \;\; l_j : C \\ \text{R}^{-} & \text{if } \phi = Rinv \\ \text{C} & \text{if } \phi = l_i : C(x) \end{cases}$$

Further examples of the DL translations assigned by our grammar to natural language sentences are shown in Table 1. Semantically, the grammar makes use of the following DL constructs: $\top$ (the most general concept), disjunction, conjunction, negation, role inverse, universal and existential restrictions. Syntactically, the grammar covers simple and complex SIDPs (i.e., SIDP consisting of more than one clause). Note that temporal or spatial relations such as *after* and *near* are not given any special semantics. They are simply DL roles i.e., binary relations. Also numerical restrictions have not been modelled yet and should be added for a finer grained semantics of nominal phrases (E.g., *at least 3 cables*).

Dissociating Grammar and Lexicon. Figure 2 shows an FB-LTAG in which trees and semantic representations are *lexicalised* in that each tree is associated with lexical items and with a specific semantic predicate. In practice though, grammar and lexicons are kept separate and the grammar contains trees and semantic schemas which are instantiated during parsing or generation using a lexicon. Figure 4 shows an illustrating example with a lexical entry on the left and the corresponding grammar unit on the right. During generation/parsing, the semantic literals listed in the lexicon (here, *Use*

[2] D^I is the domain of interpretation.



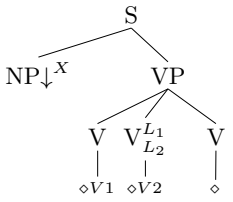
L_6:*Pipe(X)* L_0 :*subset*(L_6,L_4) L_4:*not*(L_5)
L_5:*exists*(*useArg2inv*,L_3) L_3:*Use(Z)*

$$Pipe \sqsubseteq \neg\exists useArg2^{-}.(Use)$$

Fig. 3. Derived Tree. The flat semantics representation produced by the grammar is equivalent to the Description Logic Formula shown .

and *useArg2inv*) are used to instantiate the variables (here, *A2* and *Rel*) in the semantic schema (here, L_0:*subset(X,*L_1*)* L_2:*exists(A2,*L_3*)* L_3:*Rel(Y)*). Similarly, the Anchor value (*used*) is used to label the terminal node marked with the anchor sign ($\diamond$) and each coanchor is used to label the terminal node with corresponding name. Thus, the strings *shall* and *be* will be used to label the terminal nodes $V1$ and $V2$ respectively.

Semantics:
Rel = *Use*
A2 = *useArg2inv*
Tree: nx0V
Anchor: *used*
Coanchor: V1 → *shall/V*
Coanchor: V2 → *be/V*



L_0:*subset(X,*L_1*)*
L_2:*exists(A2,*L_3*)*
L_3:*Rel(Y)*

Fig. 4. Example Lexical Entry and Grammar Unit

Importantly, this separation between grammar and lexicon supports modularity in that e.g., different lexicons and/or grammars could be plugged into the system.

In essence, the lexicon provides a mapping between natural language phrases and (simple or complex) DL concepts. For the work presented here, we built the lexicon in a rather ad hoc fashion by applying regular expressions and a customised NP chunker[3] to extract verbal and nominal lexical entries from SIDPs. This lexicon could instead

[3] We use the NLTK regular expression chunker.

be built in a more principled way e.g., by using tools for automatically identifying and extracting concepts and relations (cf. e.g., [6, 5, 4, 19]).

Similarly, to parse text whose syntax and semantics differs from those of SIDPs, another grammar could be used and mapping SIDPs to a semantic representation language other than OWL DL could be done by simply modifying the semantic component of the grammar.

3.2 Semantic Parser and Surface Realiser

Given a grammar G, a lexicon L and an input sentence S, the *semantic parser* derives from the input sentence S, the parse tree and the description logic formula associated by G and L to S. For a given sentence, the automatically extracted lexicon can produce a very large number of derivations. The parser uses a CKY algorithm [9] augmented with a simple heuristic to prune the initial search space (lexical entries whose co-anchors are not present in the input string are not selected) and a robustness mechanism for skipping unknown words, i.e. words present in the input that are absent from the lexicon.

Conversely, the *surface realiser* takes as input a grammar G, a lexicon L and a DL formula ϕ and outputs a sentence S associated by G and L to ϕ. The differences with the semantic parser are twofold. First, grammar trees are selected based on their associated semantics (rather than their associated lexical items for parsing). Second, tree combinations are not constrained by word order (during parsing, only trees whose yield matches the linear order of the input string are tried out for combination). We use the GenI surface realiser[4], a tabular bottom-up surface realisation algorithm optimised with polarity filtering[5] to map DL formulae to natural language.

4 Evaluation and Results

We evaluate our approach on a dataset of 960 System Installation Design Principles (SIDP) provided by Airbus. This evaluation is driven by three main research objectives:

- to study the coverage and robustness of the semantic parsing module (Section 4.1)
- to assess the correctness of the derived DL formulae (Section 4.2)
- to analyse the impact of our semantic parsing on the ontology learning task (Section 4.3).

In average, each SIDP sentence consists of 19.88 tokens (min: 5, max: 87). While their syntax is relatively simple (as illustrated in Example 1, an SIDP usually consists of a main clause which may be complemented with a subordinate clause expressing a condition), SIDPs have a complex compositional semantics resulting from the interaction of word order, logical operators, sentence structure and functor arity. Table 1 shows some of the semantic patterns that need to be derived. To capture this semantic

[4] See [8] for a more detailed definition of the GenI Surface Realiser

[5] Polarity filtering filters the initial search space by eliminating all combinations of grammar trees which cannot possibly lead to a successful derivation either because it can be calculated that a given tree will not be able to combine with the other trees (a resource will not be used) or, conversely, that some tree(s) are missing to yield a syntactically valid sentence (a resource will be missing).

Logical Operators	
Only S shall be used by O	$\neg S \sqsubseteq \neg\exists useA2^{-}.(Use \sqcap \exists by.O)$
S should be used by all O	$O \sqsubseteq \exists by^{-}.(Use \sqcap \exists useA2.S)$
S shall not be used by O	$S \sqsubseteq \neg\exists useA2^{-}.(Use \sqcap \exists by.O)$
Word Order	
S shall be used by O only	$S \sqsubseteq \neg\exists useA2^{-}.(Use \sqcap \exists by.\neg O)$
All S shall be used by O	$S \sqsubseteq \exists useA2^{-}.(Use \sqcap \exists by.O)$
Arity	
S shall be used	$S \sqsubseteq \exists useA2^{-}.(Use)$
S shall be used by O	$S \sqsubseteq \exists useA2^{-}.(Use \sqcap \exists by.O)$
S shall be used by O on PO	$S \sqsubseteq \exists useA2^{-}.(Use \sqcap \exists by.O \sqcap \exists on.PO)$
Sentence Structure	
S shall be used by O before entering connections	$(Use \sqcap \exists useA2.S \sqcap \exists by.O) \sqsubseteq \exists before.(Enter \sqcap \exists enterA2.Connections)$
Modifiers	
S shall be used directly by O	$S \sqsubseteq \exists useA2^{-}.(Use \sqcap \exists directly.(\exists by.O))$
S shall be used by O between C and D	$S \sqsubseteq \exists useA2^{-}.(Use \sqcap \exists useA3.(O \sqcap \exists betweenA1^{-}.(Between \sqcap \exists betweenA2.C \sqcap \exists betweenA3.D)))$

Table 1. Text and Meaning Variations

variability, we manually specify a grammar consisting of 52 trees. As mentioned above, the lexicon (10 781 lexical entries) associating word and terms (sequences of tokens) to grammar units is constructed automatically by applying regular expressions and NP chunking to the input SIDPs.

4.1 Mapping SIDPs to Complex Axioms

	Full Parse	Partial Parse	Failure	Total & Ratio
Simple SIDP	155	290	11	456 (47.50 %)
Complex SIDP	48	443	13	504 (52.50 %)
Total & Ratio	203 (21.15 %)	733 (76.35 %)	24 (2.50 %)	960 (100 %)

Table 2. Parsing Results (Coverage)

Using the grammar, the lexicon and the parser described in the preceding sections, we obtain the results shown in Table 2. Recall that simple SIDPs are rules consisting of a single clause (1a) while complex SIDPs are rules including a condition, usually expressed by a subordinate or an infinitival clause (1b). Table 2 shows that most (97.50%) SIDPs can be parsed. Manual examination of the results reveals that the few cases where parsing fails (2.50 %) are mainly due to missing lexical entries resulting in a syntactically incomplete parse tree (typically, a verbal argument is missing because the corresponding lexical entry is missing).

Table 2 also shows that a high proportion (76.35%) of the parses are partial parses that is, parses where some of the input words are ignored. Partial parses may be more or less partial though. A partial parse may simply ignore (skip over) a single word or it may ignore a whole subordinate clause. It is thus important to have some criteria for evaluating the correctness of the semantic representations derived by the parser. We show how this issue can be addressed in the following section.

4.2 Assessing Correctness

One key difficulty when using semantic parsing for ontology enrichment is that there is no known metrics for automatically checking the correctness of the semantic representations derived by the parser.

Checking Syntactic Well-Formedness. It is possible however to check the *well-formedness* of the semantic representations. If the semantic representation derived by the parser fails to convert to a well-formed description logic formula, we know that either an incorrect semantics has been assigned to some lexical entry or there is an inconsistency in the output of the syntactic component. To impose this well-formedness check, we first convert the flat semantic representations output by the parser to OWL functional syntax as explained in Section 3.1. We then input the resulting OWL formulae to the OWL Functional Syntax parser provided by OWL API 4.1.3 [10] and check that they are subclass axioms (because SIDP are rules which, in our modelling, translate to DL axioms of the form $C_1 \sqsubseteq C_2$). The results are as follows.

Full Parses		Partial Parses		All Parses	
well-formed	ill-formed	well-formed	ill-formed	well-formed	ill-formed
203	0	695	38	898	38
100%	0%	94.8%	5.2%	96%	4%

Table 3. Well-Formedness Results

These results show that most parses produce a subclass axiom for both full and partial parses. While this does not guarantee that the derived semantics correctly captures the meaning of the input SIDP, the well-formedness check allows us to quickly identify parses which are definitely incorrect. These are few (4% of all parses) and closer examination of the data reveals that these are mostly due to syntactically complex SIDP such as (3) which are insufficiently covered by the grammar and/or the lexicon.

(3) Pipes shall be defined by considering red zones for repair in order to allow the removal of the channel without having to modify the rest of the setup or to use specific procedures and tools.[6]

In sum, the well-formedness check allows us to quickly identify cases where semantic parsing yields a definitely incorrect semantic representation.

[6] Because the SIDPs we are working on are confidential data, we modified the lexical items contained in that example. The syntactic structure was preserved however and illustrates the type of syntactic examples our grammar fails to cover.

Comparing Input and Re-generated Text. We can further evaluate the correctness of full and partial parses by exploiting the fact that the grammar is declarative and can therefore be used both for parsing and for generation. Given an input SIDP S with derived semantics ϕ, we input ϕ to an existing FB-LTAG surface realiser (namely, GenI [8]) and we compare the sentence generated by this surface realiser to the initial input S. To measure the degree to which the re-generated sentence resembles the input sentence, we use BLEU [16], a precision metric standardly used in Natural Language Processing (in particular, in machine translation) for assessing the similarity between two sentences. By re-generating from partial parse semantics and comparing the resulting text with the input using a sentence similarity metrics, we can get a more precise assessment of the quality of the semantic parser output – a low *BLEU score* suggests that indeed the partial parse is very partial and that the derived semantics is likely incorrect while a high one will point to examples where e.g., a single word has been skipped. Table 4 shows the results.

		Low	Medium	High	Total (Ratio)
F-Parse	S-SIDP	0	0	155	155 (16.55%)
	C-SIDP	0	0	48	48 (5.12%)
P-Parse	S-SIDP	105	122	63	290 (30.98%)
	C-SIDP	296	102	45	443 (47.32%)
Total (Ratio)		401 (42.84%)	224 (23.93%)	311 (33.22%)	936 (100%)

Table 4. Measuring the similarity between input and re-generated sentences. Low: BLEU $\leq$ 32%, Medium: 33% $\geq$ BLEU $\leq$ 66% , High: BLEU $\geq$ 67%

For full parses, the BLEU scores are high thereby indicating that re-generated sentences are either identical or very similar to the input SIDP and suggesting that the derived DL formulae adequately capture the meaning of the input SIDP (since regenerating from it produces a sentence identical or highly similar to it). Note that, because the grammar captures some paraphrastic relations (e.g., *X will be used only with Y / X will be used with Y only*), a re-generated sentence may be different from the input SIDP even if the parse is complete.

For partial parses, the proportion of low BLEU scores is noticeably higher for complex SIDPs reflecting the fact that the syntax and semantics of conditions is only partially covered by the grammar. Low BLEU scores for simple SIDPs with partial parses are mainly due to missing or incorrect lexical entries. For instance, our lexicon does not include lexical entries for references to tables and figures. Hence sentence (4a) yields the partial parse covering the words in (4b).

(4) a. En6049 split conduit shall be attached to open backshell as defined in figure below.
b. En6049 split conduit shall be attached to open backshell.

4.3 Ontology Enrichment

Given a DL formula produced by the semantic parser, we go on to enrich an existing ontology from Airbus with that formula. The Airbus ontology describes plane components

and encompasses about 650 classes, 1300 individuals, 200 object and data properties, and more than 7400 various logical axioms. To enrich the ontology, we consider the following subtasks:

1. Identifying classes and properties contained in the DL formula which already exist in the ontology;
2. Enriching the ontology with classes and properties contained in the derived DL formulae and which do not exist in the ontology;
3. Checking for consistency and for unsatisfiable classes when adding the full DL formulae derived through parsing.

Once the ontology enrichment module[7] receives a new formula along with the lists of classes and properties it contains, the following steps are performed (see Figure 5).

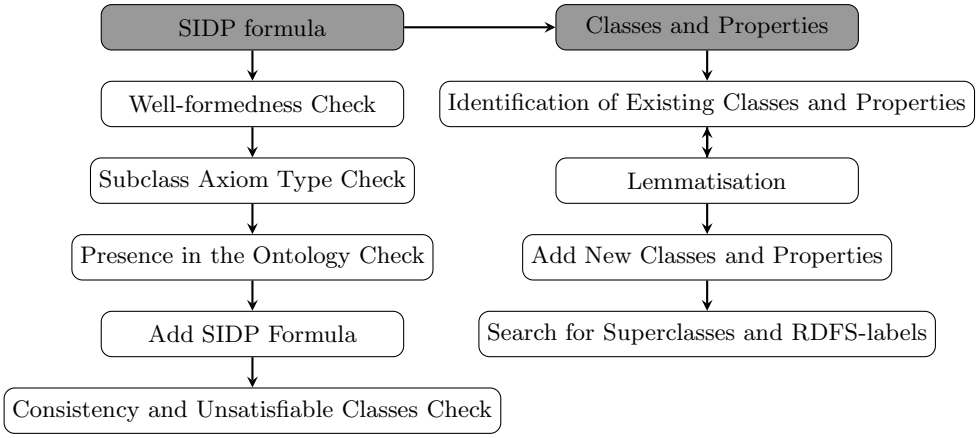


Fig. 5. Ontology enrichment component.

Classes and Properties. First, we check if classes presented in the formula exist in the ontology. As classes do not have IRIs in formulae after the semantic parsing, an IRI must be prepended to it. Along with that, we normalise a string underlying the class name: it is set to the lower-case letters, and punctuation signs are removed. Then, by a complete enumeration of all the IRIs existing in the ontology, we try to establish a match between a new class and existing classes. If an exact match was found, we consider that the class used in the axiom is already present in the ontology and should therefore not be added to it.

If the class was not found in the ontology, we resort to lemmatisation. We use the Stanford CoreNLP Toolkit [15] to lemmatise and POS-tag each token making up the class names. We also remove determiners such as *the, a* which might be present in the class names produced by the parser. For the latter, we make use of the morphological tag *DT* with which tokens were labelled during the POS-tagging phase. For instance,

[7] For interaction with the ontology, we use OWL API 4.1.3 and HermiT 1.3.8.

the green pipes becomes a class with the name *Green_Pipe*. Once the class name derived by parsing has been lemmatised, we proceed to compare it to the set of existing classes in the ontology. If a match is detected, we mark the class as an existing one. Otherwise, the class is listed as a new class, and it is added to the ontology using the base IRI, the class name being its lemmatised form.

New classes	935
Existing classes	89
New object properties	84
Existing object properties	0
superclasses found	498
RDFS-label matches found	7
new added SIDP formulae	798 (85.3%)
rejected SIDP formulae due to syntax errors	38 (4.0%)
rejected SIDP formulae due to redundancy	91 (9.7%)
rejected SIDP formulae due to inconsistency	9 (1.0%)

Table 5. Ontology Enrichment Statistics

We also apply the matching procedure used for class names to the list of properties contained in the DL formulae produced by parsing.

Statistics of the class and property identification are shown in the first section of Table 5. The number of existing classes (89 cases) we found is relatively small compared to the amount of new classes (935 cases). To better link those new classes with the existing concepts, we searched (i) for RDFS-labels which bear the same name as the derived class and (ii) for possible superclasses. We hypothesise that a class C is a superclass of another class C_i if C contains a substring of C_i. Using this strategy with superclasses was successful: we managed to relate more than a half of the new classes to their super classes (498 cases). Conversely, RDFS-labels matching did not yield many links (7 cases).

Complex DL Axioms. Once we added new classes and properties, we can enrich the ontology with the full formulae derived through parsing. Before adding a formula to the ontology, we check for well-formedness, redundancy, inconsistency and un-satisfiability:

- If the formula does not follow the OWL syntax and it is not of the subclass axiom type, it is rejected (cf. Section 4.2).
- At each step of the ontology enrichment, we refresh the list of axioms which are currently in the ontology. If the axiom is already present, we do not add it.
- If the axiom was successfully added, we perform the ontology consistency check and verify that the ontology does not have unsatisfiable classes.

The results of the ontology enrichment procedure are presented in the third section of Table 5. In total, 14.7% of formulae were rejected due to various reasons. Some of the formulae (4.0%) were not well-formed (see Section 4.2), some of them were already present in the ontology (9.7%), others were rejected as they led to inconsistency or

unsatisfiable classes (1.0%). Mainly the formula rejection occurred with partial parses. Commonly, only the main clause of an SIDP rule was parsed, leaving out the subordinate part. In such a way, different rules (*S shall be used when Y, S shall be used if Z*) were either reduced to the same partial parse (*S shall be used*), or, in case of a negative sentence, to two parses with contrasting meaning (*S shall be used, S shall not be used*). The former accounts for redundant formulae, the latter for inconsistent formulae.

Since we check the consistency and search for unsatisfiable classes after each addition of a new SIDP formula, it enables us to identify inconsistent formulae on the fly. Thus, a strong point of our approach is the immediate detection of incompatible SIDP rules without having to compute justifications, which is a challenging issue for real-world ontologies (see [11], for example).

5 Conclusion

In industries, governmental services and European organizations, system requirements are key information which need to be queried and checked for consistency. Mostly however, these are listed in documents and disconnected from formal models. Being able to translate text to model would permit e.g., to check consistency (analysis). Conversely, being able to translate models to text (generation) would allow technical authors to update the documentation to reflect changes in the model. In this paper, we showed how techniques from natural language processing could be used to provide a reversible framework for mapping natural language requirements to description logic and vice versa. We further argued that such a reversible framework helps address one key issue for semantic parsing namely, how to detect incorrect output.

In future work, we plan to explore how our reversible framework could be used to build a text-semantics corpus using data extension and recombination as suggested in [12]; and how to use this corpus to train a more robust, more generic semantic parser for system requirements.

Acknowledgments. This research has been partially supported by the ITEA 2 Eureka Cluster Programme under the ModelWriter project (Grant 13028).

References

1. Bernstein, A., Kaufmann, E., Göhring, A., Kiefer, C.: Querying ontologies: A controlled english interface for end-users. In: International Semantic Web Conference. pp. 112–126. Springer (2005)
2. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., et al.: KAON – towards a large scale Semantic Web. In: E-Commerce and Web Technologies, pp. 304–313. Springer (2002)
3. Buitelaar, P., Cimiano, P., Magnini, B.: Ontology learning from text: methods, evaluation and applications, vol. 123. IOS press (2005)
4. Charlet, J., Szulman, S., Pierra, G., Nadah, N., Téguiak, H.V., Aussenac-Gilles, N., Nazarenko, A.: Dafoe: A multimodel and multimethod platform for building domain ontologies. 2e Journées Francophones sur les Ontologies, Lyon, France: ACM (2008)
5. Cimiano, P., Völker, J.: Text2Onto : A Framework for Ontology Learning and Data-driven Change Discover. In: Natural language processing and information systems, pp. 227–238. Springer (2005)

6. Frantzi, K.T., Ananiadou, S., Tsujii, J.: The c-value/nc-value method of automatic recognition for multi-word terms. In: Research and advanced technology for digital libraries, pp. 585–604. Springer (1998)
7. Gardent, C., Kallmeyer, L.: Semantic construction in feature-based TAG. In: Proceedings of EACL. pp. 123–130. Association for Computational Linguistics (2003)
8. Gardent, C., Kow, E.: A symbolic approach to near-deterministic surface realisation using tree adjoining grammar. In: Proceedings of ACL. pp. 328–335. ACL (2007)
9. Harrison, M.A.: Introduction to formal language theory. Addison-Wesley Longman Publishing Co., Inc. (1978)
10. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL Ontologies. Semantic Web 2(1), 11–21 (2011)
11. Horridge, M., Parsia, B., Sattler, U.: Explaining inconsistencies in OWL ontologies. In: International Conference on Scalable Uncertainty Management. pp. 124–137. Springer (2009)
12. Jia, R., Liang, P.: Data recombination for neural semantic parsing. arXiv preprint arXiv:1606.03622 (2016)
13. Kaljurand, K., Fuchs, N.E.: Verbalizing OWL in Attempto Controlled English. In: OWLED. vol. 258 (2007)
14. Maedche, A., Staab, S.: Semi-automatic engineering of ontologies from text. In: Proceedings of the 12th international conference on software engineering and knowledge engineering. pp. 231–239. Citeseer (2000)
15. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: ACL (System Demonstrations). pp. 55–60 (2014)
16. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of ACL. pp. 311–318. ACL (2002)
17. Petrucci, G., Ghidini, C., Rospocher, M.: Ontology Learning in the Deep. In: Knowledge Engineering and Knowledge Management - 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings. pp. 480–495. Springer (2016)
18. Ruiz-Casado, M., Alfonseca, E., Castells, P.: Automatic Extraction of Semantic Relationships for Wordnet by Means of Pattern Learning from Wikipedia. In: International Conference on Application of Natural Language to Information Systems. pp. 67–79. Springer (2005)
19. Szulman, S., Aussenac-Gilles, N., Charlet, J., Nazarenko, A., Sardet, E., Teguiak, H.: DAFOE: A Platform for Building Ontologies from Texts. In: EKAW (2010)
20. Tablan, V., Polajnar, T., Cunningham, H., Bontcheva, K.: User-friendly ontology authoring using a controlled language. In: Proceedings of LREC (2006)
21. Völker, J., Hitzler, P., Cimiano, P.: Acquisition of OWL DL axioms from lexical resources. In: European Semantic Web Conference. pp. 670–685. Springer (2007)
22. Wang, Y., Berant, J., Liang, P.: Building a Semantic Parser Overnight. In: Proceedings of ACL 2015. pp. 1332–1342 (2015)
23. Zouaq, A., Nkambou, R.: Building domain ontologies from text for educational purposes. IEEE Transactions on Learning Technologies 1(1), 49–62 (2008)