Semanticizing syntactic patterns in NLP processing using SPARQL-DL queries

Nicola Vitucci, Mario Arrigoni Neri, Roberto Tedesco, Giuseppina Gini

Politecnico di Milano - Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
{vitucci, arrigoni, tedesco, gini}@elet.polimi.it

Abstract. Some recent works on natural language semantic parsing make use of syntax and semantics together using different combination models. In our work we attempt to use SPARQL-DL as an interface between syntactic information given by the Stanford statistical parser (namely part-of-speech tagged text and typed dependency representation) and semantic information obtained from the FrameNet database. We use SPARQL-DL queries to check the presence of syntactic patterns within a sentence and identify their role as frame elements. The choice of SPARQL-DL is due to its usage as a common reference language for semantic applications and its high expressivity, which let rules to be generalized exploiting the inference capabilities of the underlying reasoner.

1 Introduction

The tools available nowadays for natural language processing can achieve very good results on many complex tasks such as the parsing of a sentence. The limitations of such tools are in the richness of their output: the Stanford parser[1], for example, can produce a parse tree, a POS tagging and a dependency graph, but it does not provide any clues about the meaning and the topic nor it uses any semantic information to check that the sentence is semantically correct. On the other hand, projects such as FrameNet [2] aim to add some semantics to their lexical resources using the concept of *semantic frame*, intended as "a script-like conceptual structure that describes a particular type of situation, object, or event along with its participants and props", but they can only be used as references.

Several approaches have been proposed to bridge the gap, using probabilistic models [4] or a toolchain of independent components [8], using parse trees [5] or dependency graphs [1] or integrating different sources of information [10]; in this work we propose the use of SPARQL-DL[2] and the OWL API[3] as an interface between a statistical parser (i.e., the Stanford parser) and a lexical semantically-enriched resource (i.e., the FrameNet database) in order to obtain a semantic parse of a sentence.

[1] http://nlp.stanford.edu/software/lex-parser.shtml (see [7] for further details)
[2] http://www.derivo.de/en/resources/sparql-dl-api.html (see [11] for further details)
[3] http://owlapi.sourceforge.net/

The paper is organized as follows: in Section 2 we describe the sources of information we use; in Section 3 we describe the mapping between the different sources, while in Section 4 we show the whole process of a sentence analysis; in Section 5 we summarize our work and propose some future extensions.

2 Information sources

2.1 Information obtained from the Stanford parser

The Stanford parser can provide several types of output, such as part-of-speech (POS) tagging [9], typed dependencies [6] and parse tree of a sentence. The POS tags capture the role of each word in a sentence, while typed dependencies capture grammatical relations between different words; the parse tree, which we are not currently using, shows the grammatical structure of the whole sentence. The dependencies form a taxonomy: for instance, a dependency such as *dobj* (describing the *direct object* of a verb) is a specification of the dependency *obj* (a generic *object*). In the following we will speak of a dependency *graph* even if, strictly speaking, dependencies can be represented as a tree, an acyclic or a cyclic graph depending on the use of collapsed dependencies, propagation etc. (see the Stanford typed dependencies manual for more information).
As an example, parsing the sentence *I like him as a fellow* we get as a result:

```
I/PRP like/VBP him/PRP as/IN a/DT fellow/NN ./.

nsubj(like-2, I-1)        root(ROOT-0, like-2)
dobj(like-2, him-3)       prep(like-2, as-4)
det(fellow-6, a-5)        pobj(as-4, fellow-6)
```

In the first output each word is tagged with a part-of-speech (POS) tag, which captures its role in the sentence (e.g., `PRP` means that the tagged word is a personal pronoun, while `VBP` stands for "non-3rd person singular present verb"); in the second output typed dependencies are shown (e.g., the `nsubj` tag means that the word *I*, having position 1 in the sentence, is the nominal subject for the phrase governed by the verb *like* found in position 2).

2.2 Information obtained from FrameNet

The most important role within FrameNet is played by *lexical units* (*LUs*), which can be considered as "frame-evoking words" in the sense that they are used within an event or situation captured by a frame, having a specific role and meaning; for instance, the LU *like.v* is related to the use of the verb *to like* within the frame *Experiencer focus* (it only appears in one frame). Every LU, which usually is a verb, comes in a variety of different *syntactic realizations* (*SRs*), which means that it can be used in different ways within a sentence depending on its role and form (e.g., *I like apples* and *I like climbing mountains*). Every type of realization is presented as a *valence pattern*, a combination of concepts pertaining

to the frame called *frame elements* (*FEs*) such as `Experiencer`, `Reason` and `Content`, associated to syntactic realizations.
An example of valence pattern from FrameNet is:

`NP.Obj`$_{Content}$ `NP.Ext`$_{Experiencer}$ `PP[as].Dep`$_{Parameter}$

where the subscripts are the *FEs* and the words separated by a dot are called respectively *phrase type* (*PT*) and *grammatical function* (*GF*), capturing respectively the “shape” and the function of a specific FE in a sentence.
In the previous example, the sentence tagged with frame elements would be:

`[I]`$_{Experiencer}$ `like` `[him]`$_{Content}$ `[as a fellow]`$_{Parameter}$.

For the lexical unit *like.v*, other examples of valence patterns would be:

`NP.Obj`$_{Content}$ `NP.Ext`$_{Experiencer}$ `PP[for].Dep`$_{Reason}$
`[I]`$_{Experiencer}$ `like` `[the rounds]`$_{Content}$ `[for their versatility]`$_{Reason}$

`PP[at].Dep`$_{Circumstances}$ `NP.Obj`$_{Content}$ `NP.Ext`$_{Experiencer}$
`[They]`$_{Experiencer}$ `liked` `[the play]`$_{Content}$ `[at court]`$_{Circumstances}$

`NP.Ext`$_{Content}$ `AVP.Dep`$_{Degree}$ `PP[by].Dep`$_{Experiencer}$
`[This man]`$_{Content}$ `was` `[very much]`$_{Degree}$ `liked` `[by the Masai]`$_{Experiencer}$

A phrase type such as `PP[at]`, for example, means that the corresponding FE should be a phrase containing a preposition (in this case *at*); a grammatical function such as `Obj`, instead, means that the corresponding FE has an object function according to FrameNet specification (i.e. not necessarily according to its syntactic position within the sentence). In our approach, we always use these two descriptors together as a pair.

3 Mapping

3.1 Representation of a sentence

First of all, the sentence to analyze is converted to a temporary ontology where the individuals are the words in the sentence along with their position, a datatype property called *lemma* is the word itself in its base form, the classes are the POS tags and the roles are the dependency types; we use a pre-defined ontology containing all the possible POS tags and dependency types as they have been defined in the related papers (see [9] and [6]), where we added some levels of hierarchy stating, for example, that all the forms of a verb are subclasses of the class `VB`, i.e. of a generic verb; from this ontology we delete and create only the individuals every time a new sentence is analyzed. Differently from [3] we do not generalize words using their related synsets obtained from WordNet; we instead exploit their associated POS tags, the reason being a higher degree of generality of a <PT, GF> pair (i.e., we can extract different syntactic realizations of

the `PP[as].Dep` pair without the need to be bound to any specific verb). This approach also has the advantage of helping in the detection of parsing errors, when for instance a verb in the *-ing* form is tagged as an adjective.
In the previous example we would have:

```
Class(I-1, PRP)            ObjectProperty(like-2, nsubj, I-1)
Class(like-2, VBP)         ObjectProperty(ROOT-0, root, like-2)
Class(him-3, PRP)          ObjectProperty(like-2, dobj, him-3)
Class(as-4, IN)            ObjectProperty(like-2, prep, as-4)
Class(a-5, DT)             ObjectProperty(fellow-6, det, a-5)
Class(fellow-6, NN)        ObjectProperty(as-4, pobj, fellow-6)
```

from which we derive:

```
ObjectProperty(VBP, nsubj, PRP)    ObjectProperty(ROOT, root, VBP)
ObjectProperty(VBP, dobj, PRP)     ObjectProperty(VBP, prep, IN)
ObjectProperty(NN, det, DT)        ObjectProperty(IN, pobj, NN)
```

Now we can select the object properties containing only the words tagged as `PP[as].Dep`, thus obtaining a rule stating that a `PP[as].Dep` pair is characterized by the presence of the roles:

```
ObjectProperty(VBP, prep, IN)      ObjectProperty(IN, pobj, NN)
```

where the individual belonging to the class `IN` (in this case `as-4`) must have the string `as` as the value of the datatype property `lemma`(this is useful because the Stanford parser associates most of the prepositions with the same POS tag `IN`). The pattern given above can be used to check whether a sentence contains the given <PT, GF> pair (and, consequently, its related FE). In the ontology of POS tags and roles we have defined a taxonomy of POS tags as well, so rules can be easily generalized using a reasoner; in the example above the class `VBP` (a verb in non-3rd person singular present) is actually a subclass of `VB` (a verb in its base form), so the first relation can be replaced by `ObjectProperty(VB, prep, IN)` (if the presence of a verb in non-3rd person singular present is found not to be mandatory for the specific pattern). This approach is especially useful when no rules matching a sentence can be found, in that it makes it possible to loosen some constraints (e.g., it makes it possible to look for a verb in its base form when a verb in the 3rd person form is expected).

3.2 Building the matching queries

In order to match a pattern in a sentence, we represent it as a SPARQL-DL query. For example, the pattern

$\texttt{NP.Obj}_{Content}$ $\texttt{NP.Ext}_{Experiencer}$ $\texttt{PP[as].Dep}_{Parameter}$

can be represented by the SPARQL-DL query

```
PREFIX : <ontology prefix>
SELECT ?n1 ?n2 ?n3
WHERE { Type(?n1, :VB), Type(?n2, :IN), Type(?n3, :NN),
        PropertyValue(?n1, :prep, ?n2),
        PropertyValue(?n2, :pobj, ?n3),
        PropertyValue(?n2, :lemma, "as") }
```

The reason why we use SPARQL-DL to perform queries relies, as we have anticipated, on its intrinsic use of the reasoner: the query above will match all the forms of a verb and all the types of noun (singular, mass, plural, proper). Every query contains some additional information such as the name of the represented <PT, GF> pair, the number of matches over the whole database etc. The SPARQL-DL implementation we are using relies on the OWL API and covers all of OWL 2.

The same procedure is applied for all the pairs <PT, GF> present in the FrameNet database, with every query having as the first bound variable the verb evoked by the related LU (we cannot show here other examples because of space limits). In order to derive the queries, we make some hypotheses:

– every SR pair <PT, GF> is independent from the verbs it is used with, which means that it makes sense to talk of a “general” pair (e.g. `PP[by].Dep` for *like* has the same realization for *tell* in the sentences *I am liked by her* and *I was told by him*);
– we exclude the cases of null instantiation, where a frame element is not represented by any word in the sentence (e.g., in the sentence *Molly rarely eats alone* the verb *eat* is used intransitively while it is usually expected to have an object).

The queries are then built in a semi-automatic manner:

1. the pair <PT, GF> to analyze (e.g. `PP[as].Dep`) is chosen;
2. a search in FrameNet is performed to find all the examples of sentences whose SR contains a `PP[as].Dep` element (and whose related LU is a verb);
3. an example sentence is selected and parsed;
4. the elements tagged with the chosen pair are selected and shown together with their typed dependencies;
5. a SPARQL-DL query matching such dependencies is written by hand;
6. the obtained query is checked on all the previously found example sentences to find the number of matches;
7. the procedure is repeated to increase the matches or to extract a query for a different pair.

We opted out for such approach because of several limitations in the FrameNet data:

– some sentences cannot be used as examples either because they are misspelt (thus causing parsing errors) or because the parser provides wrong POS tags (e.g. a verb in *-ing* form tagged as an adjective);

– in some sentences a chunk which is tagged with a single FE can be very long, so it can be matched by several queries related to different FEs;
– it is not always trivial to decide which is the head word in a tagged FE within a sentence, especially in the case of long chunks;
– FrameNet is not a statistically representative resource.

4 Analysis process

After the preparation process, the steps which are performed to analyze and semantically parse a sentence are:

1. parsing of the sentence to obtain the POS tags, the dependency graph and the verbs;
2. conversion of the sentence in a temporary ontology;
3. execution of all the available queries and collection of all the results;
4. extraction of the valence patterns for all the frames related to the verbs;
5. check of the matching valence patterns;
6. tagging of the sentence elements with the found FEs.

The match of a valence pattern is performed by first selecting only the satisfied queries whose first bound variable (the verb in its base form) is the same one of the considered LU. Some patterns have multiple FEs with the same syntactic realization within the same pattern; this is usually not a problem as they also have the same semantic tag. To help the use and retrieval of the shape of the valence patterns, we built an ontology of FrameNet syntactic realizations; differently from other ontologies related to the same domain, our ontology contains all the syntactic realizations for every frame related to each verb. This makes it easy and unambigous to derive the possible syntactic realizations for a frame without the need of having the whole FrameNet database (to which it is anyway linked by links to the XML files containing the related LUs).

As an example of the use of this procedure we considered the *like.v* LU. The verb *like* is regular, so its base form can be easily retrieved; anyway, as we will see in the last section, this condition is not restrictive. The queries we are using are not exhaustive, meaning that they do not cover all the possible syntactic realizations.

Going back to the previous example, the queries matching the sentence *I like him as a fellow* are the ones associated to the pairs `AVP.Dep`, `NP.Ext`, `NP.Obj` and `PP[as].Dep`; as there are no other verbs in the sentence, all of them are selected. The valence patterns which are found to match this set of results are

$\texttt{NP.Obj}_{Content}$ $\texttt{NP.Ext}_{Experiencer}$ $\texttt{PP[as].Dep}_{Parameter}$

and

$\texttt{NP.Obj}_{Content}$ $\texttt{NP.Ext}_{Experiencer}$

In this case the reason of the ambiguity is given by the inclusion of the second valence pattern in the first one. After finding the matching valence patterns, the sentence is tagged accordingly; currently we only tag the head words (and the words following a preposition, in the case of prepositional phrases), so in the first case we would obtain:

`[I]`$_{Experiencer}$ `like [him]`$_{Content}$ `[as fellow]`$_{Parameter}$.

We ran this analysis on all the example sentences within the *like.v* LU, and the results we obtained show a greater difficulty in retrieving FEs whose syntactic realization is `NP.Obj` (noun phrase used as object), `NP.Ext` (noun phrase used as external argument) and `AVP.Dep` (adverb phrase); this is due mostly to the high variability in syntactic realizations (or, equivalently, to its ambiguous definition), while in prepositional phrases the prepositions act as anchors. In general, at this stage of the work, the failure rate in recognizing a FE goes between 10% and 50% (for the most difficult elements). Some more examples of results are:

`I quite like nursery rows sometimes.`
`[I]`$_{Experiencer}$ `[quite]`$_{Degree}$ `like [rows]`$_{Content}$ `[sometimes]`$_{Degree}$

`Would you like cheese in your sandwich?`
`Would [you]`$_{Experiencer}$ `like [cheese]`$_{Content}$ `[in sandwich]`$_{Content}$

The average time for the whole analysis on a modern laptop (Intel® Core^{TM}i7 2.00 GHz, 4 GB of RAM) is about 0.22 seconds.

We still have not run our analysis on databases other than FrameNet because there are no ones having comparable annotations; anyway, as an example of the errors which can be revealed by such approach, the sentence *You like this don't you?* could not be tagged because of the missing comma between *this* and *don't*, which makes the parser give a wrong output.

5 Summary and future work

In this work we have shown an application of SPARQL-DL query language to natural language processing, more specifically as a rule engine to use within a semantic parser. We have shown that the use of such formalism for this task has several advantages such as the straightforward conversion of a typed dependency graph in an ontology, the clarity of the queries (which can be written in a standard way instead of being defined using a custom language) and their generality (due to the use of a reasoner).

We are currently investigating the use of statistic measures to help in the disambiguation, that is when more than one pattern match the same sentence; our approach includes the use of the frequencies of each pattern and the probability for each FE within a pattern to have a certain position within the pattern itself. Also, we are adopting a lemmatizer to retrieve the base form of verbs after they have been found in the analyzed sentence; this lets it possible to deal with irregular verbs.

As future extensions we plan to add a semantic expansion/disambiguation module, in order both to cope with synonyms which are not explicitly present in the FrameNet database and to deal with semantically wrong sentences (e.g. to put in evidence problems in sentences like *I ride a pencil* when the pencil is not a rideable object); several authors use other databases such as WordNet and VerbNet, while we are planning to use OpenCyc as well. Another extension is related to a fully automatic generation of the queries to use as matching rules: there are several problems to be solved, mainly depending on the ambiguity of the tags which make it difficult to automatically derive the head word of every FE, and on parsing or typing errors.

References

1. P. Adolphs, F. Xu, H. Li, and H. Uszkoreit. Dependency graphs as a generic interface between parsers and relation extraction rule learning. In *Proceedings of the 34th Annual German conference on Advances in Artificial Intelligence*, KI'11, pages 50–62, Berlin, Heidelberg, 2011. Springer-Verlag.
2. C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet Project. In *Proc. of the 36th Annual Meeting of the Association for Computational Linguistics and 17th Intl. Conf. on Computational Linguistics - Volume 1*, ACL '98, pages 86–90, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.
3. R. Basili, D. Croce, D. D. Cao, and C. Giannone. Learning semantic roles for ontology patterns. In *Web Intelligence/IAT Workshops*, pages 291–294, 2009.
4. D. Chen, N. Schneider, D. Das, and N. A. Smith. SEMAFOR: Frame argument resolution with log-linear models. In *Proc. of the 5th Intl. Workshop on Semantic Evaluation*, SemEval '10, pages 264–267, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
5. Y. S. Choi. Tree pattern expression for extracting information from syntactically parsed text corpora. *Data Min. Knowl. Discov.*, 22:211–231, January 2011.
6. M.-c. De Marneffe, B. Maccartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *In LREC 2006*, 2006.
7. M.-C. de Marneffe and C. D. Manning. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser '08, pages 1–8, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
8. K. Erk and S. Padó. SHALMANESER - A Toolchain For Shallow Semantic Parsing. In *Proceedings of LREC 2006*, 2006.
9. B. Santorini. Part-Of-Speech tagging guidelines for the Penn Treebank project (3rd revision, 2nd printing). Technical report, Department of Linguistics, University of Pennsylvania, Philadelphia, PA, USA, 1990.
10. L. Shi and R. Mihalcea. Putting pieces together: combining FrameNet, VerbNet and WordNet for robust semantic parsing. In *Proceedings of the 6th international conference on Computational Linguistics and Intelligent Text Processing*, CICLing'05, pages 100–111, Berlin, Heidelberg, 2005. Springer-Verlag.
11. E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *In 3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.