Extracting RDF triples using the Stanford Parser

Yassine Hamoudi
Ecole Normale Supérieure de Lyon
France
yassine.hamoudi@ens-lyon.fr

Tom Cornebize
Ecole Normale Supérieure de Lyon
France
tom.cornebize@ens-lyon.fr

Abstract

One of the most important tasks in *Natural Language Processing* (NLP) theory consists of simplifying sentences in order to extract their structure and the most relevant information they contain. This paper presents a method to obtain RDF triples (subject-predicate-object) from questions using the Stanford Parser. Our approach is mainly based on a careful analysis of the dependency tree produced by the Stanford Parser. We will also make use of some extra tools (named-entity recognizer, part-of-speech tagger, etc). Our algorithms have been implemented and are part of a query answering tool[1].

1 Introduction

The most common way to find answers on the Internet is to enter keywords in a web search engine (Google, Baidu, Yahoo!, etc) and to browse the returned results. However, in some cases the user is looking for a short and precise answer (e.g. *"capital of France"*, *"birth date of Obama"*). Instead of having to navigate through many web pages, he could hope to directly obtain the answer whenever it is possible.

Question Answering over Linked Data theory tries to tackle this problem. It consists in all the techniques and algorithms used to automatically answer questions asked in natural language (e.g. *"What is the capital of France?"*). The core of the process is to map the questions asked by the users into **normal forms** that can be easily handled by databases-querying tools. The Resource Description Framework (RDF) provides a natural normal form that consists in sets of subject-predicate-object triples. For instance, the sentence *"Paris is the capital of France."* can be represented by the triple: (France, capital, Paris).

Triples with holes stand for the missing information required by the question. Thus, the expected normal form of *"What is the capital of France?"* is: (France, capital, ?).

A natural way to map questions into normal forms is to analyse the grammatical structure of the sentence. Some algorithms use the concrete syntax tree (or parse tree) representation to perform this task (Rusu et al., 2007; Defazio, 2009). Another popular representation is the dependency tree. This structure describes grammatical relationships between the words of the sentence. Although the dependency tree is often used in Question Answering (Zouaq et al., 2006; Zouaq et al., 2007), algorithms to produce RDF triples from it are pretty rare and poorly described.

The aim of this paper is to provide a full algorithm that maps questions into RDF triples, mainly using the dependency tree representation output by the Stanford Parser (de Marneffe and Manning, 2013). We will focus on factoid *wh*-questions. This work was carried out in collaboration with *Projet Pensées Profondes* (Deep Thought Project), a project conducted by seven students that aims to propose a natural language question answering framework[1] (a demo is available online).

2 Methods

The normal form is obtained by applying some operations on the dependency tree output by the Stanford Parser. We detail throughout this section the different steps of our algorithm and we apply them on the example:

Where was the president of the United States born?

[1] http://projetpp.github.io/

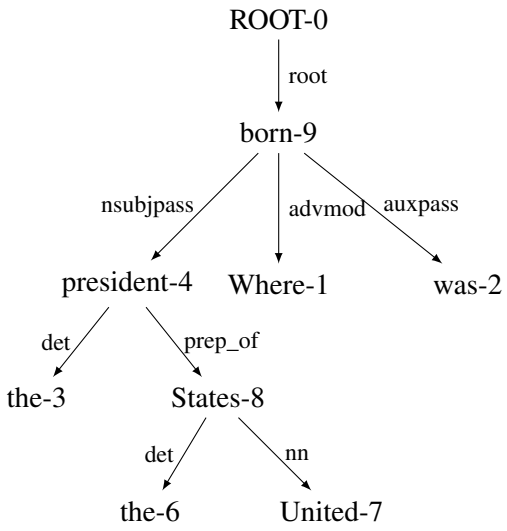


Figure 1: Dependency tree of *Where was the president of the United States born?*

2.1 Stanford dependency tree

The Stanford Parser[2] is a natural language parser developed by the *Stanford Natural Language Processing Group*. It is well-documented and considered as a "state of the art" program. The Stanford Parser provides classical tools of NLP theory (part-of-speech tagger, named-entity recognizer, constituency parser, etc.) but also a very powerful dependency parser.

A dependency tree reflects the grammatical relationships between words in a sentence. Figure 1 provides an overview of such a tree in our example. For instance, the edge:

$$\texttt{president} \xrightarrow{\texttt{det}} \texttt{the}$$

means that *the* is a determiner for *president*.

The Stanford typed dependencies manual (de Marneffe and Manning, 2013) details the full list and description of possible grammatical dependencies.

2.2 Normal form

Our representation of questions into normal form is based on the RDF format. Each sentence is represented by a *tree* composed of subject-predicate-object triples and operators (union, intersection, conjunction, etc). An exhaustive presentation of our data model is available online[3]. We expose succinctly its main points below.

[2] `http://nlp.stanford.edu/software/corenlp.shtml`

[3] `https://github.com/ProjetPP/Documentation/blob/master/data-model.md`

Leaves of the trees are **values** among:

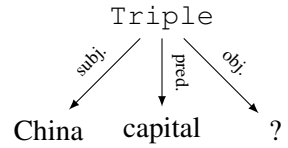
- *resource*: represents an entity in the universe. It may be a person, a location, a date, etc.
- *list*: an ordered collection of resources without duplicates. A list with only one element can be represented by the element itself ([Foo] may be written Foo).
- *missing*, denoted ?: an unknown value that needs to be found to answer the question.

Internal nodes are operators among:

- *triple*, denoted (a, b, c) or `Triple`(a, b, c) (where a,b, and c are *lists*): represents a relation between a, b and c, where a is the subject, b the predicate and c the object. Each triple has exactly one *missing*. For instance, the triple (George Washington, birth date, ?) associates to *George Washington* and *birth date* the list of resources that satisfy the relation.
- *union* and *intersection*, denoted $\cup$ and $\cap$: takes two lists as input and outputs the union, or intersection, of them.
- *sort*, denoted `Sort`: `Sort`(l, a) sorts the *list* l in increasing order according to the predicate a.
- *first* and *last*, denoted `First` and `Last`: takes one list as input and outputs the first, or last, element of it.

Our data model actually includes other values and operators (especially boolean) that are not fully supported yet in the following algorithms.

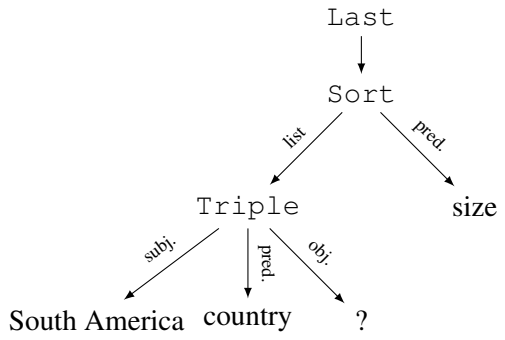
Figures 2, 3 and 4 provide some examples of normal forms that we expect to produce for different factoid *wh*-questions.



Linear representation:

(China, capital, ?)

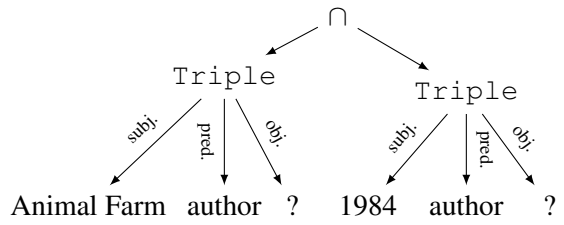
Figure 2: Possible normal form for *What is the capital of China?*



Linear representation:

`Last(Sort((South America, country, ?), size))`

Figure 3: Possible normal form for *What is the biggest country in South America?*



Linear representation:

(Animal Farm, author, ?) ∩ (1984, author, ?)

Figure 4: Possible normal form for *Who is the author of "Animal Farm" and "1984"?*

A possible normal form for *Where was the president of the United States born?* is presented in figure 5. Its linear representation is: ((United States, president, ?), birth date, ?).

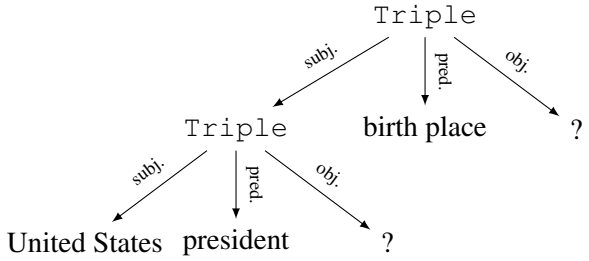


Figure 5: Possible normal form for *Where was the president of the United States born?*

2.3 Dependency tree simplification

The simplification of the dependency tree consists of several operations applied on the tree to change its shape and prepare the production of the normal form. At the end of the simplification, all the dependency tags (the Stanford Parser uses more than 50 grammatical dependency tags) have been replaced by a small subset of eight (new) tags.

Preprocessing

First of all, we perform multiword expressions recognition in order to merge all the nodes of the tree that belong to a same expression. We merge especially neighbour nodes with a same named-entity tag (for instance, *United* and *States* are tagged `LOCATION` and therefore merged into *United States*). Other kinds of merging are performed in part 2.3, using the grammatical dependencies.

Then, the question word (Who, Where, How tall, etc.) is identified and removed from the dependency tree (it will be used later to build and improve the normal form). This operation is performed using a list of about 30 question words and looking at the two first words of the sentence (they are supposed to contain the question word).

Finally, lemmatization is applied on nouns (e.g. *presidents* is mapped to *president*) and nounification on verbs (e.g. *born* is mapped to *birth*).

Preprocessing is illustrated in figure 6.

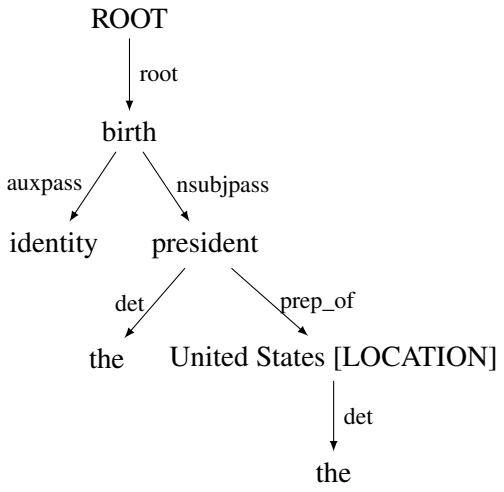


Figure 6: Preprocessed dependency tree of *Where was the president of the United States born?*

Global transformations

Two kinds of transformations modify the global shape of the tree. These operations are applied for `amod` dependencies if the output node of the dependency edge is an ordinal or a superlative (we look at its named-entity and part-of-speech tags).

They are also applied for `conj` dependencies.

Global transformations add a new node and rebalance the tree. Figure 7 illustrates the transformation applied for conjunction dependencies (`conj_or`, `conj_and`, etc.). Figure 8 gives the transformation for an `amod` dependency (the part-of-speech tag of *highest* is JJS).

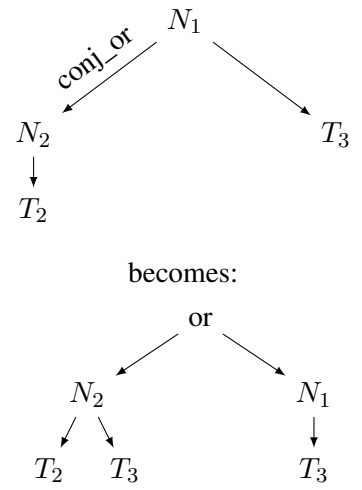


Figure 7: Remove `conj_or` dependencies

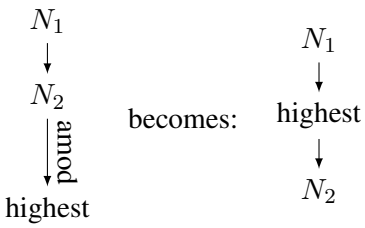


Figure 8: Remove `amod` dependencies

Local transformations

Finally, all remaining edges are analysed locally. We apply one of the following rule to each of them, depending on their dependency tags:

- **Merge** the two endpoints of the edge. It is a new step of multiword expressions merging. This rule is applied, among others, for `nn` or remaining `amod` dependencies. For example:

  `birth` $\xrightarrow{\texttt{nn}}$ `date` becomes:

  `birth date`

- **Remove** the sub tree pointed out by the edge. This rule is applied for `det` dependencies for example:

  `president` $\xrightarrow{\texttt{det}}$ `the` becomes:

  `president`

- **Replace** the dependency tag by a *triple production tag*. We have currently defined eight different types of *triple production tags*: $R_0, \cdots, R_5, R_{spl}, R_{conj}$. These tags are used to produce the final result. We attribute the same tag to dependency relationships that must produce the same type of nodes or connectors in the normal form. For instance, we replace `prep` and `poss` dependencies by the tag R_5: `president` $\xrightarrow{\texttt{prep\_of}}$ `France` becomes `president` $\xrightarrow{R_5}$ `France`.

Finally, depending on the question word we add some information in certain nodes (for instance, if the question word is *where* we try to add the word *place* into the child of the root of the tree). This extra transformation is supported actually for about 30 question words.

Figure 9 illustrates the dependency tree simplification applied on our example.

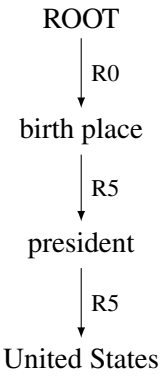


Figure 9: Simplified dependency tree of *Where was the president of the United States born?*

2.4 Construction of the normal form

The final step of the algorithm maps the tree obtained at the end of part 2.3 to a normal form that follows the data model presented in part 2.2. The transformation is a recursive function `Normalize` that takes a tree T as input and outputs the normal form. We give the main rules applied to obtain `Normalize`(T). Trees are denoted by $T_{...}$ and nodes by $N_{...}$. For a tree T (resp. a node N), $\underline{T}$ (resp. $\underline{N}$) represents the words contained in the root of T (resp. in N).

First of all, if N is a leaf then $\texttt{Normalize}(N) = \underline{N}$ (*value* node).

Then, rules R_0 to R_5 are used to produce the different types of possible triples. Table 1 gives $\texttt{Normalize}(N \xrightarrow{R_i} T)$ when T is the only child of N and $R_i \in \{R_0, \cdots, R_5\}$. For instance, $\texttt{Normalize}(N \xrightarrow{R_3} T)$ is $\texttt{Triple}(?, \underline{N}, \texttt{Normalize}(T))$.

Rule R	$\texttt{Normalize}(N \xrightarrow{R} T)$
R_0	$\texttt{Normalize}(T)$
R_1	$\underline{T}$
R_2	$\texttt{Triple}(\underline{T}, \underline{N}, ?)$ if T is a leaf
	$\texttt{Normalize}(T)$ otherwise
R_3	$\texttt{Triple}(?, \underline{N}, \texttt{Normalize}(T))$
R_4	$\texttt{Triple}(?, \texttt{Normalize}(T), \underline{N})$
R_5	$\texttt{Triple}(\texttt{Normalize}(T), \underline{N}, ?)$

Table 1: Normalization rules for $R0, \cdots, R_5$

When the root N has more than one child, all linked by a rule in $\{R_0, \cdots, R_5\}$, we split the tree to use table 1. This transformation is depicted in figure 10.

$$\texttt{Normalize}\left(\begin{array}{ccc} & N & \\ \swarrow^{R_{i_1}} & \downarrow & \searrow^{R_{i_n}} \\ T_{i_1} & \cdots & T_{i_n} \end{array}\right) =$$

$$\texttt{Normalize}(N \xrightarrow{R_{i_1}} T_{i_1}) \cap \cdots$$
$$\cdots \cap \texttt{Normalize}(N \xrightarrow{R_{i_n}} T_{i_n})$$

Figure 10: Normalization rule for $R0, \cdots, R_5$ $(R_{i_1}, \cdots, R_{i_n} \in \{R_0, \cdots, R_5\})$

Rules R_{spl} and R_{conj} are produced by the global transformations performed in part 2.3. When rule R_{spl} occurs, the root node contains a superlative or an ordinal and it has only one child. Depending on the superlative/ordinal (we have listed the most common ones), `Normalize` outputs the relevant connector nodes. A general example is presented in figure 11. Rule R_{conj} is used for conjunction. A general example is presented in figure 12.

$$\texttt{Normalize}\left(\text{biggest} \xrightarrow{R_{spl}} T\right) =$$

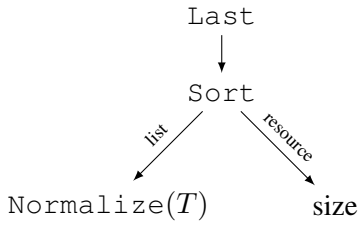


Figure 11: Normalization rule for R_{spl}

$$\texttt{Normalize}\left(\begin{array}{ccc} & R_{conj} \quad \text{and} \quad R_{conj} & \\ T_1 & & T_2 \end{array}\right) =$$

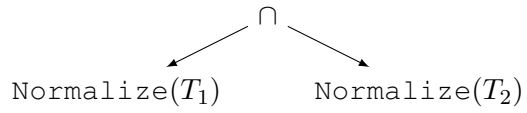


Figure 12: Normalization rule for R_{conj}

All the rules we use are available in our implementation[4]. After applying all of our transformations we obtain for our example the normal form that was predicted in figure 5.

3 Results

The previous algorithm has been implemented in Python 3. We use the collapsed dependency tree output by the *CoreNLP* parser with the flag `-makeCopulaHead`. We access *CoreNLP* using a Python wrapper[5] we have patched to support Python 3. The code is available online[4]. It includes a documentation and demo files that allow the user to quickly test the algorithm. Moreover, a query answering tool using our algorithm is in development[6]. When the user enters a question, he can get the answer and visualize the normal form by clicking on the `Show internal results` button.

We have displayed in appendix A four normal forms produced by our algorithm on questions

[4] https://github.com/ProjetPP/PPP-QuestionParsing-Grammatical

[5] https://bitbucket.org/ProgVal/corenlp-python/overview

[6] http://askplatyp.us/

used in TREC-8 contest. These results are quite representative of what our tool can do.

4 Discussion

Our goal was to extract subject-predicate-object triples from questions using the grammatical dependency tree output by the Stanford Parser. The results we obtain are close to the expected normal forms. Moreover, we have noticed that our algorithm also supports some *nominal sentences* such as *"capital of England"*. These results prove the relevance of studying the dependency tree for extracting RDF triples. Indeed, it can be seen that the structure of the dependency tree is very close to the structure of the normal form we build. The constituency tree (as in (Rusu et al., 2007; Defazio, 2009)) does not allow to handle as complex sentences as we do.

As we mentioned earlier, our algorithm is used into a question answering framework available online[6]. The normal form we produced is handled by a module performing queries on Wikidata. Furthermore, our normal form could also be mapped to SPARQL queries. Naturally, questions with complex grammatical structures are more likely to be misparsed. We also do not currently support all kinds of questions (Yes/No questions for instance).

Future work will consist in improving the different parts of the algorithm (multiword expressions recognition, better analysis of grammatical dependencies). We also plan to train the Stanford Parser on our own annotated data set. Moreover, our approach can be easily transposed to languages using similar grammatical dependencies than English. Finally, the extraction of RDF triples is useful into many fields of NLP theory, other than Question Answering. For instance, simplifying a sentence into a structured normal form is an important task of automatic text summarization or speech processing.

Acknowledgements

This paper was made possible thanks to the help and work from members of the *Projet Pensées Profondes*: Marc Chevalier, Quentin Cormier, Raphaël Charrondière, Thomas Pellissier Tanon, Tom Cornebize, Valentin Lorentz and Eddy Caron (adviser).

References

Marie-Catherine de Marneffe and Christopher D. Manning, 2013. *Stanford typed dependencies manual*. `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

Aaron Defazio. 2009. Natural language question answering over triple knowledge bases. `http://users.cecs.anu.edu.au/~adefazio/TripleQuestionAnswering-adefazio.pdf`.

Delia Rusu, Lorand Dali, Blaž Fortuna, Marko Grobelnik, and Dunja Mladenić. 2007. Triplet extraction from sentences. `http://ailab.ijs.si/delia_rusu/Papers/is_2007.pdf`.

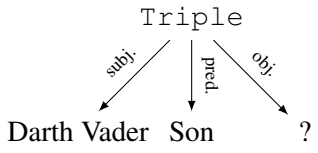
Amal Zouaq, Roger Nkambou, and Claude Frasson. 2006. The knowledge puzzle: An integrated approach of intelligent tutoring systems and knowledge management. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '06, pages 575–582, Washington, DC, USA. IEEE Computer Society.

Amal Zouaq, Roger Nkambou, and Claude Frasson. 2007. Building domain ontologies from text for educational purposes. In *Proceedings of the Second European Conference on Technology Enhanced Learning: Creating New Learning Experiences on a Global Scale*, EC-TEL'07, pages 393–407, Berlin, Heidelberg. Springer-Verlag.

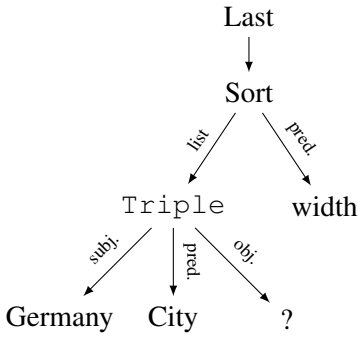
Appendices

A Example of normal forms produced by the algorithm

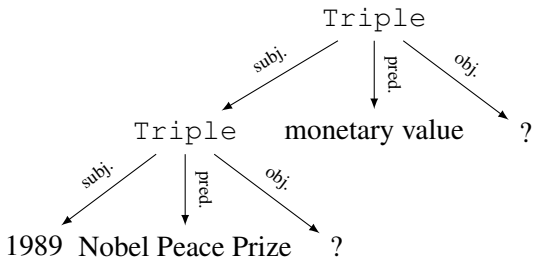
Who was Darth Vader's son?



What is the largest city in Germany?



What was the monetary value of the Nobel Peace Prize in 1989?



What is the continent of Fiji and Guam?