TRIPLET EXTRACTION FROM SENTENCES

Delia Rusu, Lorand Dali*, Blaž Fortuna°, Marko Grobelnik°, Dunja Mladenić°*
* Technical University of Cluj-Napoca, Faculty of Automation and Computer Science
G. Barițiu 26-28, 400027 Cluj-Napoca, Romania
° Department of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel: +386 1 477 3127; fax: +386 1 477 3315
E-mail: delia.rusu@gmail.com, loranddali@yahoo.com
{blaz.fortuna, marko.grobelnik, dunja.mladenic}@ijs.si

ABSTRACT

In this paper we present an approach to extracting subject-predicate-object triplets from English sentences. To begin with, four different well known syntactical parsers for English are used for generating parse trees from the sentences, followed by extraction of triplets from the parse trees using parser dependent techniques.

1. INTRODUCTION

According to the approach presented in [13], we define a triplet in a sentence as a relation between subject and object, the relation being the predicate. The aim here is to extract sets of the form {subject, predicate, object} out of syntactically parsed sentences, with four parsers, namely Stanford Parser, OpenNLP, Link Parser and Minipar. The work presented in [13] used proprietary parser that already provided the logical form of a sentence including subject, object, predicate information, while our work is based on using publicly available parsers. Stanford Parser and OpenNLP both generate a treebank structure, and therefore will be discussed together. Furthermore, we present the results obtained using the Link Parser application, which is based on the link grammar. Finally, we describe the triplet extraction algorithm for the parse tree given by Minipar.

The performance of the applications was measured using a system with CPU 2.80 GHz and 2.00 GB RAM. We used the same data for all measurements, meaning 100 sentences extracted from news articles.

2. TREEBANK PARSERS

A treebank is a text corpus where each sentence belonging to the corpus has a syntactic structure added to it.

Because of the common outputted parse tree of Stanford Parser and OpenNLP, we developed a similar algorithm for triplet extraction for the two parsers. In this section the algorithm will be presented in more detail

2.1. Triplet Extraction Algorithm for Treebank Parsers

A sentence (S) is represented by the parser as a tree having three children: a noun phrase (NP), a verbal phrase (VP) and the full stop (.). The root of the tree will be S.

Firstly we intend to find the subject of the sentence. In order to find it, we are going to search in the NP subtree. The subject will be found by performing breadth first search and selecting the first descendent of NP that is a noun. Nouns are found in the following subtrees:

Subtree	The type of noun found
NN	noun, common, singular or mass
NNP	noun, proper, singular
NNPS	noun, proper, plural
NNS	noun, common, plural

Secondly, for determining the predicate of the sentence, a search will be performed in the VP subtree. The deepest verb descendent of the verb phrase will give the second element of the triplet. Verbs are found in the following subtrees:

Subtree	The type of verb found
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle or gerund
VBN	verb, past participle
VBP	verb, present tense, not 3rd person singular
VBZ	verb, present tense, 3rd person singular

Thirdly, we look for objects. These can be found in three different subtrees, all siblings of the VP subtree containing the predicate. The subtrees are: PP (prepositional phrase), NP and ADJP (adjective phrase). In NP and PP we search for the first noun, while in ADJP we find the first adjective. Adjectives are found in the following subtrees:

Subtree	The type of adjective found
JJ	adjective or numeral, ordinal
JJR	adjective, comparative

JJS	adjective, superlative

```
function TRIPLET-EXTRACTION(sentence) returns a solution, or failure

    result ← EXTRACT-SUBJECT(NP_subtree)
        ∪ EXTRACT-PREDICATE(VP_subtree)
        ∪ EXTRACT-OBJECT(VP_siblings)
    if result ≠ failure then return result
    else return failure

function EXTRACT-ATTRIBUTES(word) returns a solution, or failure
    // search among the word's siblings
    if adjective(word)
        result ← all RB siblings
    else
        if noun(word)
            result ← all DT, PRP$, POS, JJ, CD, ADJP, QP, NP siblings
        else
            if verb(word)
                result ← all ADVP siblings
    // search among the word's uncles
    if noun(word) or adjective(word)
        if uncle = PP
            result ← uncle subtree
    else
        if verb(word) and (uncle = verb)
            result ← uncle subtree
    if result ≠ failure then return result
    else return failure

function EXTRACT-SUBJECT(NP_subtree) returns a solution, or failure
    subject ← first noun found in NP_subtree
    subjectAttributes ←
        EXTRACT-ATTRIBUTES(subject)
    result ← subject ∪ subjectAttributes
    if result ≠ failure then return result
    else return failure

function EXTRACT-PREDICATE(VP_subtree) returns a solution, or failure
    predicate ← deepest verb found in VP_subtree
    predicateAttributes ←
        EXTRACT-ATTRIBUTES(predicate)
    result ← predicate ∪ predicateAttributes
    if result ≠ failure then return result
    else return failure

function EXTRACT-OBJECT(VP_sbtree) returns a solution, or failure
    siblings ← find NP, PP and ADJP siblings of
        VP_subtree
    for each value in siblings do
        if value = NP or PP
            object ← first noun in value
        else
            object ← first adjective in value
        objectAttributes ←
            EXTRACT-ATTRIBUTES(object)
    result ← object ∪ objectAttributes
    if result ≠ failure then return result
    else return failure
```

Figure 1: *The algorithm for extracting triplets in treebank output.*

Moreover, for each element composing the triplet, we find its attributes (modifiers). For example, the attributes of a noun are mainly adjectives, the attributes of a verb are adverbs. Figure 1 presents the algorithm for extracting triplets of the form **subject – predicate – object**.

2.2. Stanford Parser and OpenNLP

2.2.1. Description

Stanford Parser is a natural language parser developed by Dan Klein and Christopher D. Manning from The Stanford Natural Language Processing Group [1, 2]. The package contains a Java implementation of probabilistic natural language parsers; a graphical user interface is also available, for parse tree visualization. The application we developed uses version 1.5.1, released on 11.06.2006. The software is available at [3].

OpenNLP is a collection of projects for natural language processing. We used SharpNLP, which is the C# variant. SharpNLP is distributed with the GNU lesser general public license and can be downloaded from [4].

2.2.2. Stanford Parser and OpenNLP Parse Trees

Stanford Parser generates a Treebank parse tree for the input sentence. Figure 2 depicts the parse tree for the sentence "*A rare black squirrel has become a regular visitor to a suburban garden"*. The triplet extracted out of this sentence is **squirrel – become – visitor**, as shown in Figure 2.

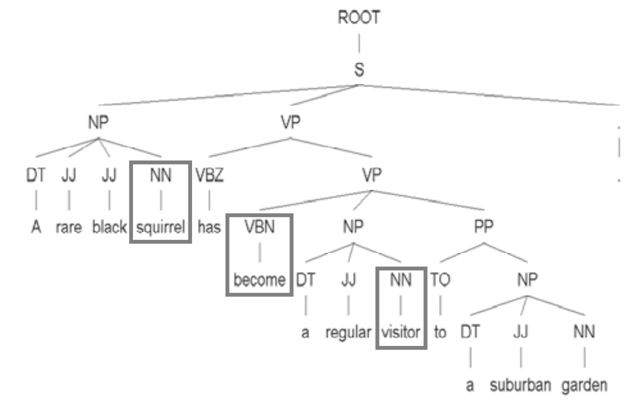

Figure 2: *The parse tree generated by Stanford Parser, when the input sentence is "A rare black squirrel has become a regular visitor to a suburban garden." The triplet extracted out of this sentence is* ***squirrel – become – visitor.***

After applying the triplet extraction algorithm presented in Figure 1, we obtain the result presented in Figure 3.
]

The subject *squirrel* has as attributes the adjectives *rare, black* and the article *a;* the word *become* has as attributes its auxiliary *has* and the object *visitor* has as attributes the adjective *regular*, the article *a* and the noun *garden* with its attributes: preposition *to*, article *a* and adjective *suburban*.

OpenNLP's parse tree is similar to that of Stanford Parser, thus we will not discuss it separately.

2.2.3. Performance

The application using Stanford Parser was written in Java. It parsed the sentences in 178.1 seconds, generating 118 triples.

The application using OpenNLP was written in C#. It parsed the sentences in 29.95 seconds, generating 168 triples.

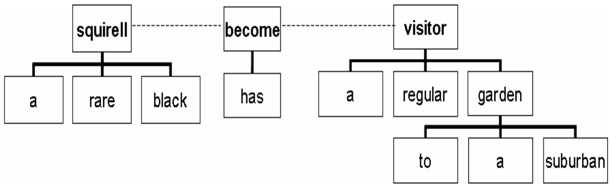

Figure 3: *The triplet structure containing the triplet elements together with their attributes.*

3. LINK PARSER

3.1. Description

This application uses the link grammar, generating a linkage after parsing a sentence [7, 8, 9, 10]. It can be downloaded from the web site [5].

For the sentence given as an example in the case of Stanford Parser, the output of Link Parser is presented in Figure 4.

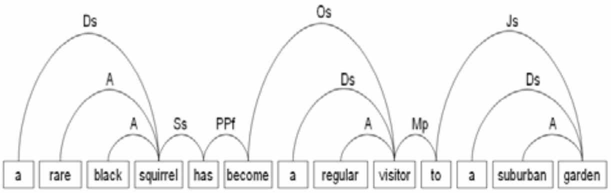

Figure 4: *The linkage obtained by parsing the sentence "A rare black squirrel has become a regular visitor to a suburban garden."*

Detailed explanations of what the different link labels mean are available at [6].

```
function EXTRACT-ATTRIBUTES(word) returns a solution, or failure
        // search among the word's siblings
        if verb(word)
                result ← all the words which have been visited on the way from the right word of the S_link to the predicatePivot ∪ adverbs connected by E_link to the left or {EBm*, N}_links to the right
        else
                if noun (word)
                        result ← all words connected by {D*, J*, A*}_links to the left and {Mp, NI*, TM, TY}_links to the right
        if result ≠ failure then return result
        else return failure

function TRIPLET-EXTRACTION(sentence) returns a solution, or failure
        determine S_link
        subjectPivot ← word left of S_link
        subjectAttributes ←
                EXTRACT-ATTRIBUTES (subjectPivot)

        currentWord ← first word right of S_link
        do
                predicatePivot ← currentWord
                currentWord ← next word to the right
        while link from predicatePivot to the right ∈ {Pv*$^1$, Pg*, PP*, I*, TO, MVi*}
        // if the verb is a phrasal verb
        if predicatePivot has a K_link to the right
                predicatePivot ← predicatePivot + word corresponding to the link
        predicateAttributes ←
                EXTRACT-ATTRIBUTES (predicatePivot)

        if link from predicatePivot ∈ {O, Os*, Op*, MVpn*}
                // noun objects
                objectPivot ← objectPivot + word corresponding to the link
        else
                if link from predicatePivot ∈ {Pa*, MVa*}.
                        // adjective objects
                        objectPivot ← objectPivot + word corresponding to the link
        // find objects connected to the predicate by prepositions
        if link from predicatePivot ∈ {MVp*, Pp*, OF, TO}
                //link from predicate to preposition
                preposition ← word corresponding to the link
                if link from preposition ∈ {J*, TI, I*, ON}
                        objectPivot ← objectPivot + word corresponding to the link
        for each value in objectPivot do
                objectAttributes ← objectAttributes + EXTRACT-ATTRIBUTES (value)

        result ← subjectPivot ∪ subjectAttributes ∪ predicatePivot ∪ predicateAttributes ∪ objectPivot ∪ objectAttributes
        if result ≠ failure then return result
        else return failure
```

Figure 5: *The algorithm for extracting triplets using Link Parser*

For a sentence many possible linkages can be found. A linkage is composed of one or more sublinkages.

If the sentence is a compound sentence with two simple sentences, then the linkage will contain two sublinkages, one for each sentence. There can also be more sublinkages if in place of some part of speech there is a list of words separated by conjunctions. For example for the sentence *"The squirrel and the elephant are in the garden."* there will be two sublinkages, one for *"The squirrel is in the garden",* and one for *"The elephant is in the garden".*

3.2. Triplet Extraction Algorithm

In each sublinkage there is exactly one link which begins with an 'S'. This link connects the subject to the predicate.

The left of the S link is the subject pivot (squirrel). If the subject pivot is linked to other words on its left by G (names of persons) or NN (parts of numerals) links, then these words will be added to the subject word (e.g. Sergio Ricardo Gomez, one hundred million). From the right of the S link the predicate will be obtained. Once the predicate is found, objects can be determined as well by following several links, as shown in Figure 5, where a pseudocode version of the algorithm is given.

The EXTRACT-ATTRIBUTES function searches for attributes of verbs and nouns. Attributes of nouns (the subject and some objects) can be to the left or to the right. In the case of the Mp link to the right, a link to a preposition is made, and from that preposition to a noun by {J*[1], I*}. The attribute will be this noun, not the preposition to which Mp links. This attribute noun will have its own attributes, one of them being the preposition which connects it to the noun it determines by the Mp link.

3.3. Performance

The application was written in C++. It parsed the sentences in 271 seconds, generating 110 triples.

4. MINIPAR

4.1. Description

Minipar is a parser developed by Dekang Lin. In the application we developed, version 0.5 for Windows was used.

4.2. Minipar Parse Tree

The parse tree generated by Minipar for the sentence given as an example for the previous parsers is described in Figure 6. *E<number>* is a label used to describe the root of a new clause. The information that one node of the parse tree contains is the following: a label that uniquely identifies the node, the word, its root form, its grammatical category and its grammatical relation.

4.3. Triplet Extraction Algorithm

In order to find the subject element of the triplet, we look for nodes in the tree having the grammatical relation *s*. The predicates will be found by identifying the nodes with grammatical relation *i* and the objects will be nodes having grammatical relation *pcomp*. In addition, attributes will be determined by examining the subtree of each of the triplet elements.

4.4. Performance

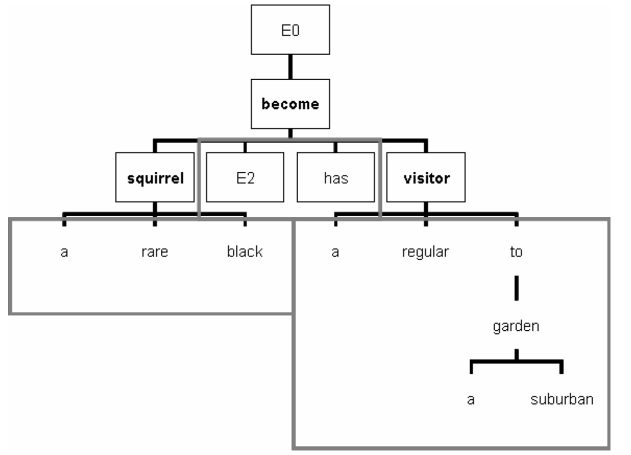

Figure 6: *The parse tree generated by Minipar for the sentence "A rare black squirrel has become a regular visitor to a suburban garden." The attributes of the triplet elements are also marked.*

The application was written in C++. It parsed the sentences in 104 seconds, generating 153 triples.

5. CONCLUSION

In this paper we presented algorithms for extracting of {subject, predicate, object} triplets from a given parse tree of a sentence.

As part of the future work we plan to use the triplets as an input for applications such as text summarization [13, 14], fact extraction [15] and *template induction* (discovering event templates in news articles) with the purpose of extracting facts for Cyc [12].

Acknowledgement

This work was supported by the Slovenian Research Agency and the IST Programme of the EC under NeOn (IST-4-027595-IP) and PASCAL (IST-2002-506778).

[1] The * means that the link has to start with the letters left to the * and can end at the * or continue with any letters.

References

[1] D. Klein, C. D. Manning. 2003. *Fast Exact Inference with a Factored Model for Natural Language Parsing*. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, Cambridge, MA: MIT Press, pp. 3-10, 2003.

[2] D. Klein, C. D. Manning. *Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pp. 423-430, 2003.

[3] Stanford Parser web page: http://nlp.stanford.edu/software/lex-parser.shtml

[4] SharpNLP: http://www.codeplex.com/sharpnlp

[5] Link Parser web page: http://www.link.cs.cmu.edu/link/.

[6] Link labels web page: http://www.link.cs.cmu.edu/link/dict/summarize-links.html

[7] D. Sleator and D. Temperley. *Parsing English with a Link Grammar*. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.

[8] D. Sleator and D. Temperley. *Parsing English with a Link Grammar. Third International Workshop on Parsing Technologies*, 1993.

[9] J. Lafferty, D. Sleator, and D. Temperley. *Grammatical Trigrams: A Probabilistic Model of Link Grammar. Proceedings of the AAAI Conference on Probabilistic Approaches to Natural Language*, October, 1992.

[10] D. Grinberg, J. Lafferty and D. Sleator. *A robust parsing algorithm for link grammars*. Carnegie Mellon University Computer Science technical report CMU-CS-95-125, and *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague, September, 1995.

[11] D. Lin, P. Pantel. *DIRT - Discovery of Inference Rules from Text*. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001*. pp. 323-328, 2001.

[12] Cyc project: http://www.cyc.com/

[13] J. Leskovec, M. Grobelnik, N. Milic-Frayling. *Learning Sub-structures of Document Semantic Graphs for Document Summarization*. In *Proceedings of the 7th International Multi-Conference Information Society IS 2004, Volume B*. pp. 18-25, 2004.

[14] J. Leskovec, N. Milic-Frayling, M. Grobelnik. Impact of Linguistic Analysis on the Semantic Graph Coverage and Learning of Document Extracts, *National Conference on Artificial Intelligence*, 2005.

[15] O. Etzioni, M. Cafarella, D. Downey, A. M. Popescu, T. Shaked, S. Soderland, D. S. Weld, A.Yates. *Unsupervised named-entity extraction from the Web: An experimental study*. Artificial Intelligence, Volume 165, Issue 1, June 2005, Pages 91-134.