Semgrex-Plus: a tool for automatic dependency-graph rewriting

Fabio Tamburini
University of Bologna
FICLIT
Bologna, Italy
fabio.tamburini@unibo.it

Abstract

This paper describes an automatic procedure, the Semgrex-Plus tool, we developed to convert dependency treebanks into different formats. It allows for the definition of formal rules for rewriting dependencies and token tags as well as an algorithm for treebank rewriting able to avoid rule interference during the conversion process. This tool is publicly available[1].

1 Introduction

Creating a treebank, annotating each sentence with its syntactic structure, is certainly a time-consuming and error prone task. For these reasons, treebanks often require maintenance and revisions to correct mistakes or to adapt it to different needs.

In big projects, such as the Universal Dependencies (UD) project (Nivre et al., 2016), guidelines updates due to new language addition, change in theoretical approaches of a specific phenomenon management, mistakes or other changes often require specific tools to automate, at the maximum possible level, the process of treebank substructures rewriting. Moreover, the treebanks developed for a specific language need often to be completely converted to adhere to other standards, for example to comply to the UD specifications and conventions.

For phrase-structure treebanks there are various tools able to perform trees rewriting, such as *Tregex/Tsurgeon* pair (Levy and Andrew, 2006), but for dependency treebanks, largely dominant in these years, no specific rewriting tool seems to be available to the community. There are some generic, though very powerful, graph rewriting tools (Guillaume et al., 2012; Ribeyre, 2013) that can be adapted to this task, but with some issues discussed in the last Section.

The StanfordNLP group developed a very interesting tool to perform treebank search by using a specialised query language. Using the *Semgrex* package[2] (Chambers et al., 2007) the user is able to specify search patterns and retrieve all the matching subgraphs inside a specific dependency treebank. This tool is very flexible and rich of operators, allowing the user to design powerful search patterns.

We extended the behaviour of this package adding some new functionalities for automatic dependency-graph rewriting useful for treebank maintenance, revision and conversion, producing a new tool, called *Semgrex-Plus*, that we made publicly available[1].

Semgrex-Plus can be used, in principle, to convert any dependency treebank represented using the CoNLL format into a different format that does not require re-tokenisation steps, or to rewrite some parts of the treebank using different dependency structures, labels and/or word tags.

This paper is organized as follows: we provide the description of the original Semgrex tool in Section 2; we then introduce the Semgrex-Plus tool describing the addition to the original tool in Section 3; in Section 4 the rule checking procedures and in Section 5 we present a treebank-conversion experiment using Semgrex-Plus; in the last Section we draw some provisional conclusion.

2 The Stanford Semgrex Search Language

Semgrex represents nodes in a dependency graph as a (non-recursive) attribute-value matrix. It then uses regular expressions for subsets of attribute values. For example,

[1] https://github.com/ftamburin/Semgrex-Plus.git

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

`{word:record;tag:/N.*/}` refers to any node that has a value 'record' for the attribute 'word' and a 'tag' starting with the letter 'N', while '`{}`' refers to any node in the graph. The most important part of Semgrex regards the possibility to specify relations between nodes or group of nodes. See Table 1 for a reference taken from the original documentation and Table 2 for some examples of Semgrex search patterns together with the retrieved subgraphs.

For example, '`{}=1 <subj=A {}=2`' finds all the pairs of nodes connected by a directed 'subj' relation in which the first node (`{}=1`) is the dependent and the other the head. Logical connectives can be used to form more complex patterns and node/relation naming (the '=' assignments) can help to retrieve matched nodes/relations from the patterns. Please refer to (Chambers et al., 2007) or to the online manual[3] for a more complete description of the Semgrex query language.

3 Semgrex-Plus

Unfortunately Semgrex is only a query language and, in its original form, cannot be used to rewrite dependency (sub)graphs. In order to extend the possibility of Semgrex, we then modified the original application to manage pairs of patterns: the first is used to search into the treebank for the required subgraphs, and the second is used to specify how the retrieved subsgraphs have to be rewritten. For example the pattern pair "`{tag:det}=1 >arg=A {tag:noun}=2`" $\rightarrow$ "`{tag:ART}=1 <DET=A {tag:NN}=2`", what we called a '*Semgrex-Plus rule*', changes the direction of the dependency between the head and the dependent and, at the same time, changes the words tags and relation label. The starting '*search*' pattern and final '*rewrite*' pattern have to contain the same number of nodes and dependency edges. Node and relation naming has been the fundamental trick to introduce such extension, allowing for nodes and relations matching between the search pattern and rewrite pattern.

3.1 Rule Application Procedure

For converting a treebank into a different format or to adjust some specific subgraphs, by applying a complex set of Semgrex-Plus rules, it is necessary to define a specific procedure in order to avoid rule application interference: the application of a rule to the treebank changes the treebank structure potentially blocking the application of the remaining rules.

The solution we adopted decouples the search and rewrite operations for the rule application. We defined a set of new rewriting operations on a general dependency treebank:

- DEL_REL(graphID, depID, headID): deletes a dependency edge between two graph nodes;
- INS_REL(graphID, depID, headID, label): inserts a new labelled dependency edge between two graph nodes;
- REN_TAG(graphID, nodeID, tag): replace the tag of a specific graph node.

The conversion task has been implemented as a three-steps process:

- first of all, each Semgrex-Plus rule is always applied to the original treebank producing a set of matching subgraphs that have to be rewritten;
- for each match, a set of specific operations for rewriting the subgraph corresponding to the processed matching are generated and stored;
- lastly, the whole set of rewriting operations produced processing the entire set of Semgrex-Plus rules, each applied to the original treebank, is sorted by graphID, duplicates are removed and every operation is applied graph by graph respecting the following order: first dependency deletions, second dependency insertions and lastly tag renaming.

This way of processing the original treebank and transforming it into the new format should guarantee that we do not experience rule interference due to the conversion procedure, because the generation of the rewriting operations due to the Semgrex-Plus rules application is decoupled from the real treebank rewriting.

Figure 1 shows the results of the application of three Semgrex-Plus rules to two simple dependency graphs.

[3]`https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/semgraph/semgrex/SemgrexPattern.html`

Symbol	Meaning
{}=1	Generic node without any attribute with ID='1'
{tag:W}=2	Generic node with attribute tag='W' and with ID='2'
A <reln=X B	A is the dep. of a rel. reln (with ID='X') with B
A >reln=X B	A is the gov. of a rel. reln (with ID='X') with B
A <<reln B	A is the dep. of a rel. reln in a chain to B following dep.->gov. paths
A >>reln B	A is the gov. of a rel. reln in a chain to B following gov.->dep. paths
A x,y<<reln B	A is the dep. of a rel. reln in a chain to B following dep.->gov. paths btw. dist. of x and y
A x,y>>reln B	A is the gov. of a rel. reln in a chain to B following gov.->dep. paths btw. dist. of x and y
A == B	A and B are the same nodes in the same graph
A . B	A is immediately precedes B, i.e. A.index() == B.index() - 1
A $+ B	B is a right immediate sibling of A
A $- B	B is a left immediate sibling of A
A $++ B	B is a right sibling of A
A $−− B	B is a left sibling of A
A @ B	A is aligned to B

Table 1: Supported node specification and relations and their symbols by the original Stanford Semgrex tool. Semgrex-Plus currently supports only the first four operators in rewriting rules.

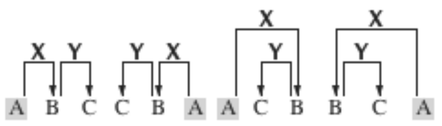
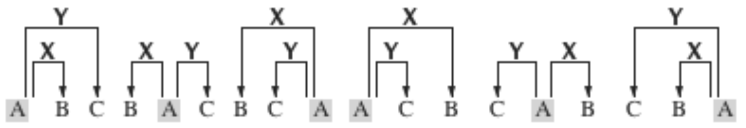
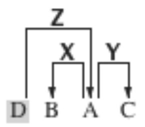
Semgrex search pattern	Retrieved subgraphs
{A} >X ({B} >Y {C})	
{A} >X {B} >Y {C}	
{D} >Z ({A} >X {B} >Y {C})	 ... See previous example to build all retrieved subgraphs.

Table 2: Some examples of Semgrex search patterns and the corresponding retrieved subgraphs.

4 Rule Overlap/Interference Checking

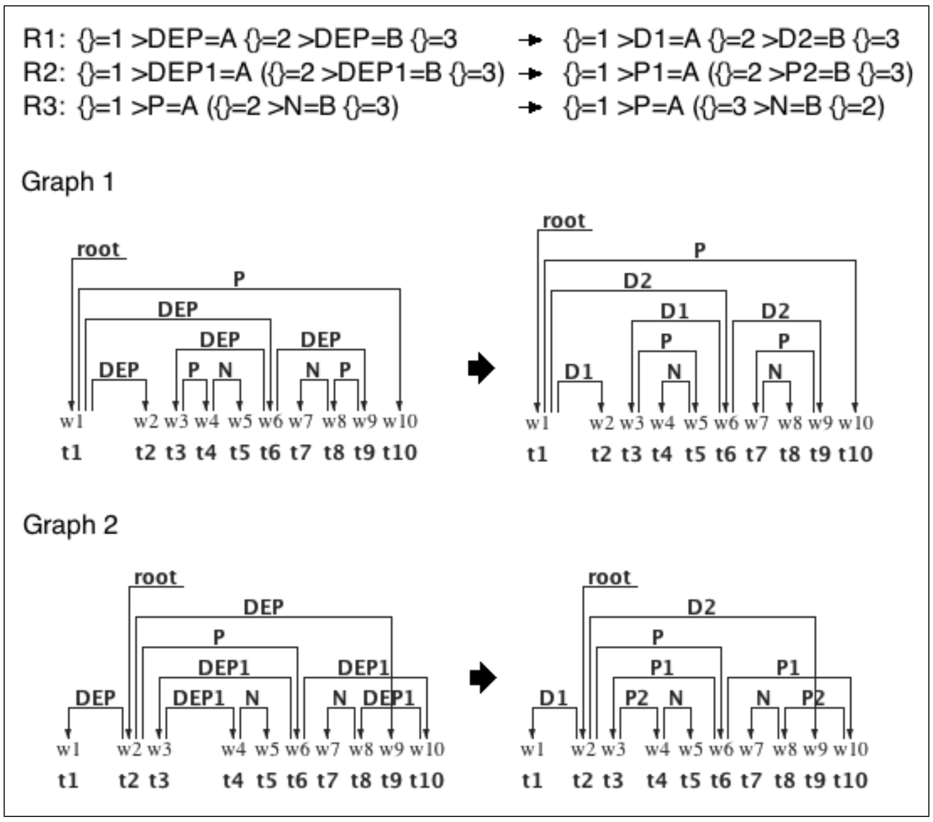
Decoupling the 'search' from 'rewrite' operations should avoid any interference artificially introduced by the conversion procedure, but do not guarantee that errors in rules definition could generate problems due to rules interference.

We designed a specific tool that compare each rule in the ruleset with all the other rules and try to find potential interference between them. In order to find this potential problems (without applying the rules to a specific treebank we do not know in advance if a problem effectively will arise or not, thus we prefer to call them 'potential') we have to check if two rules exhibit specific kinds of overlaps, but only in the subgraphs that will be actually modified by the rewrite pattern.

The first step identify which edges in the search pattern are modified by the rewrite pattern of each rule. An edge is modified by a specific rule application if:

- the relation will be modified (the relation will connect different nodes, one or both, or it will have a different label);
- one of its nodes will be modified by an attribute change.

In the second step each rule is compared to all the others by considering the intersection between the two subgraphs formed by modified edges. If the intersection is not empty and



```
R1: {}=1 >DEP=A {}=2 >DEP=B {}=3        → {}=1 >D1=A {}=2 >D2=B {}=3
R2: {}=1 >DEP1=A ({}=2 >DEP1=B {}=3)    → {}=1 >P1=A ({}=2 >P2=B {}=3)
R3: {}=1 >P=A ({}=2 >N=B {}=3)          → {}=1 >P=A ({}=3 >N=B {}=2)
```

Graph 1
root
P
DEP
DEP
DEP
DEP
P N
N P
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
root
P
D2
D1
D2
P
P
D1
N
N
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
Graph 2
root
DEP
P
DEP1
DEP1
DEP
DEP1 N
N DEP1
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
root
D2
P
P1
P1
D1
P2 N
N P2
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10

Figure 1: An example of graph conversion: the results of the application of three Semgrex-Plus rules to two simple dependency graphs.

```
R4: {}=1 >P=A ({tag:X}=2 >N=B {}=3)  → {}=1 >P=A ({tag:X}=2 >C=B {}=3)
R5: {}=1 >N=A {}=2                   → {}=1 >C=A {}=2
R6: {tag:Y}=1 >N=A {}=2              → {tag:Y}=1 >C=A {}=2
```

Figure 2: Rules added to the first three (R1-R3 in Figure 1) to demonstrate rule overlap checking.

a) the two search patterns completely match, then we have a *full overlap* between rules and this mark a problem. The rule matching is similar to a unification process thus an empty node (e.g. {}=1) will match with any other node.

b) the two search patterns do not completely match, then we got a *partial overlap* between rules and this is only a potential problem because, in principle, the two rules should apply to different subgraph without creating real issues.

An empty intersection between rules modified subgraphs do not create any problem.

If we add the rules in Figure 2 to the ones presented in Figure 1 and apply the described algorithm to check for rule overlapping, we will obtain two full overlaps for rule pairs R3-R4 and R5-R6 and three partial overlaps for rule pairs R3-R5, R3-R6 and R4-R5.

5 Some Linguistic Examples

We used an early version of the Semgrex-Plus package to automate the conversion of the Venice Italian Treebank (Delmonte et al., 2007) into a different format, namely the MIDT+ format (Bosco et al., 2012), in order to start the merging of this treebank into the Italian Universal-Dependency treebank (Alfieri and Tamburini, 2016).

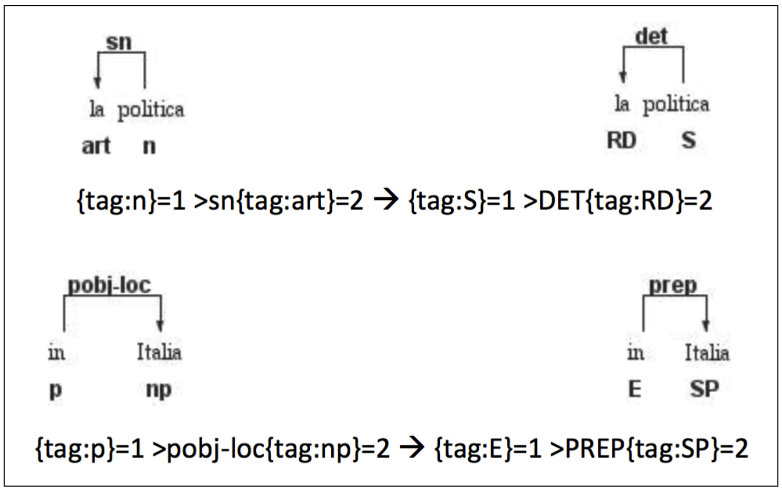

sn
la politica
art n
det
la politica
RD S
{tag:n}=1 >sn{tag:art}=2 → {tag:S}=1 >DET{tag:RD}=2
pobj-loc
in Italia
p np
prep
in Italia
E SP
{tag:p}=1 >pobj-loc{tag:np}=2 → {tag:E}=1 >PREP{tag:SP}=2

Figure 3: Some simple examples of rules that do not modify the dependency structures.

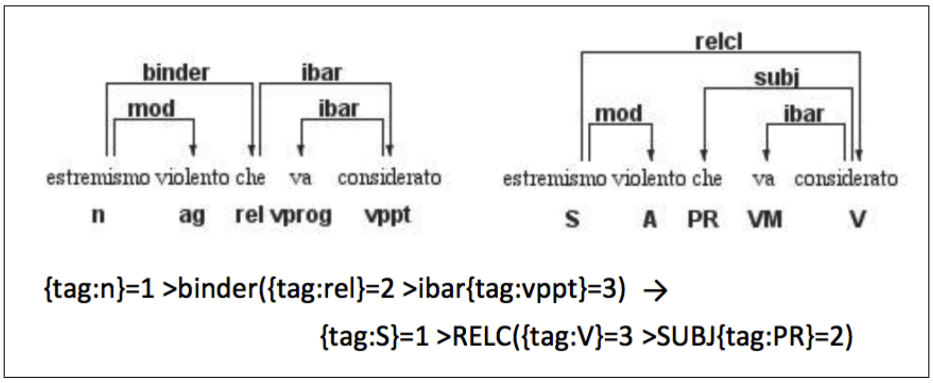

binder
mod
ibar
ibar
estremismo violento che va considerato
n ag rel vprog vppt
relcl
subj
mod
ibar
estremismo violento che va considerato
S A PR VM V
{tag:n}=1 >binder({tag:rel}=2 >ibar{tag:vppt}=3) →
{tag:S}=1 >RELC({tag:V}=3 >SUBJ{tag:PR}=2)

Figure 4: An example of rule that rewrite the dependency structure.

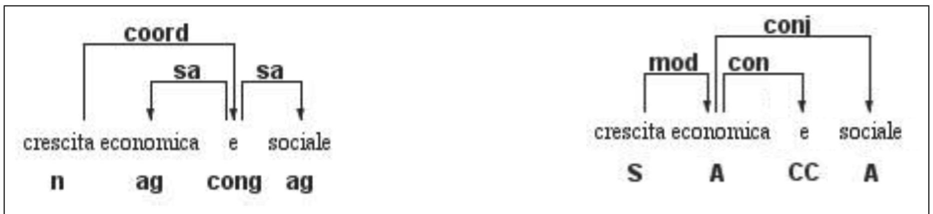
The set of rules manually written for converting VIT dependency structures can be subdivided into two macro-classes: (a) rules that do not modify the structures and (b) rules that need to modify the dependencies, both in term of edge direction and in term of different structuring between the involved nodes.

Regarding the rules that do not modify the dependency structures, they simply rename the dependency label using a 1:1 or an N:1 look-up table, as VIT, with respect to MIDT+, typically involves more specific dependency types. Figure 3 outlines some simple examples of such kind of conversions.

There are, of course, other kind of operations on subgraphs that require also the rewriting of the dependency structure. A good example concerns relative clauses in which the role of the relative pronoun and, as a consequence, the connections of the edge expressing the noun modification are completely different in the two formalisms. Figure 4 shows one example of this kind of rewriting.

Cases of coordination presented several problems for treebank conversions: in VIT the head of the coordinated structure is linked to the connective and then the two (or possibly more) coordinated structures can be linked with a wide range of different dependency types (e.g. between phrases - *sn*, *sa*, *savv*, *sq*, *sp*, predicative complements - *acomp*, *ncomp*, adjuncts - *adj*, *adjt*, *adjm*, *adjv*, subjects - *subj*, objects - *obj*, etc.) leading to a large number of different combinations. Moreover, each dependency combination has to be further specified by the different token tags. MIDT+ represents coordinate structures in a different way: the connective and the second conjunct are both linked to the first conjunct that is connected to the head of the coordinated structure.

Figure 5 shows one example: the first formal rule represents an abstract rule pattern that has to be filled with all the real tag combinations found in VIT, generating a huge number of different



```
{ }=1 >coord({ }=2 >sa{ }=3 >sa{ }=4) → { }=1>MOD({ }=3 >CON{ }=2 >CONJ{ }=4)

{tag:n}=1 >coord({tag:cong}=2 >sa{tag:ag}=3 >sa{tag:ag}=4) →

{tag:S}=1 >MOD({tag:A}=3>CON{tag:CC}=2>CONJ{tag:A}=4)
```

Figure 5: An example of coordination structures in VIT and MIDT+ and the conversion rule.

rules, one of them outlined by the second complete Semgrex-Plus rule. This process generated more than 2,800 different rules for handling all the coordinated structures in VIT.

There is also a need for a third kind of rules for rewriting single PoS-tags that might have remained unchanged during the main conversion process.

Applying all the 4,250 Semgrex-Plus rules we wrote, we obtained a converted treebank in which 228,534 out of 280,641 dependency relation were automatically converted, giving a global coverage of 81.4%.

To test the effectiveness of the conversion procedure and the conversion rules we randomly selected 100 sentences (2582 dependency relations to be converted) from the treebank and manually checked every newly created dependency relation, both in term of the connected nodes and the assigned label. We obtained the following results: among the 2008 relations that have been automatically converted we found 125 wrongly converted dependency relations. So, on this sample, we obtained a coverage of 2008/2582 = 77.8%, slightly less than on the whole treebank, with a conversion error rate = 125/2008 = 6.2%.

6 Conclusions

This paper presents the tool Semgrex-Plus, derived from the StanfordNLP Semgrex tool, we developed to allow for dependency structure rewriting inside a specific treebank.

This procedure can be, in principle, adaptable to any conversion between different dependency treebank formats or to modify the specific dependency structures, labels and/or word tags connected with a particular phenomenon.

Beside some simple examples of dependency structure rewriting using Semgrex-Plus we gave in this paper, we briefly reported on the use of this tool for automating the conversion of an Italian dependency treebank into a different format, in order to show the effectiveness of this tool when used in big and complex conversion projects.

To the best of our knowledge, there are only general-graph rewriting tools (Guillaume et al., 2012; Ribeyre, 2013) available to automatise this task for dependency graphs. Though this packages are very powerful and quite flexible, it is not clear, however, if they apply the rewriting operations when a rule pattern is found, modifying the treebank immediately, or not, and if there are some rule-checking procedures for raising potential problems in rules application, because, as we have seen in Section 4, a sequential application of the various rewriting rules could complicate the process of treebanks conversion. On the other hand, decoupling the pattern recognition from the rewriting operations, as done by the Semgrex-Plus tool, guarantee that we can write rules having in mind the original tagset without any modification, but we should still check and avoid interference among the conversion rules.

We can also find some powerful treebank converters in literature but they are usually tied to specific pair of tagsets (often tailored to the Penn treebank) (Johansson and Nugues, 2007; Choi and Palmer, 2010), and cannot be easily adapted to general needs, or are devoted to tree manipulation, for example the tool 'Tregex' (Levy and Andrew, 2006).

In any case, the formal rules for converting a treebank have to be manually written by using the proposed tool syntax and the final result has to be carefully tested to check the effectiveness of the conversion rules. The tool do not guarantee that, writing incomplete or wrong rules, the final result will be fine. For example, if we need to invert the direction of a dependency, we must include in the rule conversion also the node governing such dependency, in order to properly manage the graph and avoid the generation of illegal graphs (e.g. non-rooted trees/graphs).

References

Linda Alfieri and Fabio Tamburini. 2016. (Almost) Automatic Conversion of the Venice Italian Treebank into the Merged Italian Dependency Treebank Format. In *Proc. 3rd Italian Conference on Computational Linguistics - CLiC-IT 2016*, pages 19–23, Napoli, Italy.

Cristina Bosco, Simonetta Montemagni, and Maria Simi. 2012. Harmonization and Merging of two Italian Dependency Treebanks. In *Proc. of LREC 2012, Workshop on Language Resource Merging*, pages 23–30, Istanbul.

Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher Manning. 2007. Learning Alignments and Leveraging Natural Logic. In *Proc. of the Workshop on Textual Entailment and Paraphrasing*, pages 165–170.

Jinho Choi and Martha Palmer. 2010. Robust Constituent-to-Dependency Conversion for English. In *Proc. of 9th International Workshop on Treebanks and Linguistic Theories - TLT9*, Tartu, Estonia.

Rodolfo Delmonte, Antonella Bristot, and Sara Tonelli. 2007. VIT - Venice Italian Treebank: Syntactic and Quantitative Features. In *Proc. Sixth International Workshop on Treebanks and Linguistic Theories*.

Bruno Guillaume, Guillaume Bonfante, Paul Masson, Mathieu Morey, and Guy Perrier. 2012. Grew: un outil de réécriture de graphes pour le TAL. In Gilles Sérasset Georges Antoniadis, Hervé Blan-chon, editor, *12ième Conférence annuelle sur le Traitement Automatique des Langues (TALN'12)*, Grenoble, France. ATALA.

Richard Johansson and Pierre Nugues. 2007. Extended Constituent-to-dependency Conversion for English. In *Proc. of NODALIDA 2007*, Tartu, Estonia.

Roger Levy and Galen Andrew. 2006. Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proc. of 5th International Conference on Language Resources and Evaluation - LREC 2006*, Genoa, Italy.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666, Portorož, Slovenia.

Corentin Ribeyre. 2013. Vers un système générique de réécriture de graphes pour l'enrichissement de structures syntaxiques. In *RECITAL 2013 - 15ème Rencontre des Etudiants Chercheurs en Informatique pour le Traitement Automatique des Langues*, pages 178–191, Les Sables d'Olonne, France.