Using Constraint Grammar for Treebank Retokenization

Eckhard Bick
University of Southern Denmark
eckhard.bick@mail.dk

Abstract

This paper presents a Constraint Grammar-based method for changing the tokenization of existing annotated data, establishing standard space-based ("atomic") tokenization for corpora otherwise using MWE fusion and contraction splitting for the sake of syntactic transparency or for semantic reasons. Our method preserves ingoing and outgoing dependency arcs and allows the addition of internal tags and structure for MWEs. We discuss rule examples and evaluate the method against both a Portuguese treebank and live news text annotation.

1 Introduction

In an NLP framework, tokenization can be defined as the identification of the smallest meaningful lexical units in running text. Tokens can be both words, symbols or numerical expressions, but there is no general consensus on what constitutes a token boundary. For instance, are "instead of" or "Peter Madsen" 1 or 2 tokens? Should German "z. B." (for example) be 2 tokens and English "e.g." 1 token, just because the former contains a space? What about a word that allows optional space (*insofar* as vs. *in so far as*)? Far from being a merely theoretical issue, tokenization conventions strongly influence parsing schemes and results (e.g. Grefenstette & Tapanainen 1994). Thus, contextual rules become simple (and therefore safer) when faced with single-token names, conjunctions and prepostions rather than complex ones. Conversely, contractions such as Portuguese "na" (= em [in] a [the]) can only be assigned a meaningful syntactic analysis when split into multiple tokens, in this case allowing the second part (the article) to become part of a separate np.

Tokenization is often regarded as a necessary evil best treated by a preprocessor with an abbreviation list, but has also been subject to methodological research, e.g. related to finite-state transducers (Kaplan 2005). However, there is little research into changing the tokenization of a corpus once it has been annotated, limiting the comparability and alignment of corpora, or the evaluation of parsers. The simplest solution to this problem is making conflicting systems compatible by changing them into "atomic tokenization", where all spaces are treated as token boundaries, independently of syntactic or semantic concerns. This approach is widely used in the machine-learning (ML) community, e.g. for the Universal Dependencies initiative (McDonald et al. 2013). The method described in this paper can achieve such atomic tokenization of annotated treebank data without information loss, but it can also be used for grammar-based tokenization changes in ordinary annotation tasks, such as NER.

2 Retokenization challenges

What exactly atomic (space-based) retokenization implies, is language-dependent, and may involve both splitting and fusion of tokens, for fused multi-word expressions (MWEs) and split contractions, respectively. While the former, not least for NER, is a universal issue, the latter is rare in Germanic languages (e.g. *aren't, won't*), but common in Romance languages. In both cases, the retokenization method should conserve existing information, i.e. MWE boundaries in one case, and morphosyntactic tags of contraction parts in the other. Linguistically, token-splitting is the bigger problem, because it needs *added*

information: (a) partial POS tags, (b) additional internal dependency links and (c) new internal hook-up points for existing outgoing and incoming dependency links. Unlike simple tag conversion for, say, morphological features, this cannot be achieved with a conversion table.

3 CG retokenization

Our solution is based on two unique features of the CG3 compiler (Bick & Didriksen 2015). The first allows context-based insertion, deletion and substitution of cohorts (token + 1 or more readings), and was originally intended for spell- and grammar-checking. Thus, we implemented token fusion by either inserting a (new) fused token and then removing all original tokens, or by substituting a token with a larger, fused one containing the subsequent token (rules 1), then removing the latter (rule 2). The other feature introduces cohort splitting rules and was added specifically for retokenization. Such a rule can specify how to split a target token and manipulate its parts using regular expression matching (rule 3). In a separate rule field, a dependency chain is stipulated across the split token.

3.1 Multi-word expressions

How an MWE is to be split, obviously depends on its POS and composition. A simple case are name chains entirely made up of proper nouns. Here, (part) lemmas equal (part) tokens, and internal structure is simply a left- (or right-) leaning dependency chain. With other word classes, however, there may be inflection and complex internal structure. The Portuguese proper noun-splitting rule (1a), for instance, breaks up TARGET named entities (NE) of the type PROP+PRP+PROP (e.g. *"(Presidente do) Conselho de Administração" [Administrative Council President]*) - if necessary, iteratively. The asterisk for part 1 means that the first part inherits all tags (pos, edge label, features) from the NE as a whole, while c->p means that it also inherits incoming child (c) and outgoing parent (p) dependencies. For parts 2 and 3, independent new POS tags (PRP, PROP) and syntactic function labels (@N<, @P<) are provided. All parts receive a numbered MWE id (<MWE1>, <MWE2> etc.), and the original MWE token is retained in a separate tag (<MWE:...>. Note that the new parts may themselves be MWEs, needing further splits. Contractions contained in a NE (*do* [of the_sg_m], *pelas* .. [by the_pl_f]) need to be split (1c), in order to be treated like other, "free" contractions in the corpus. (1c) starts with a default male singular reading which is "corrected" by (1d) into female and/or plural where necessary.

Rule (1b) targets a complex adverb, *dali para diante [from here onward, from now on]*, performing not only a 3-way split on space, but also splitting the contraction *dali* (de+ali PRP+ADV). The '*' on the first part means that it will inherit form and function tags from the MWE as a whole, and "c->p" means it will also inherit both incoming (child) and outgoing (parent) dependencies.

```
(1a) SPLITCOHORT:multipart-prop (
"<$1>"v "$1"v <MWE1><MWE:$1=$2=$3>v * c->p
"<$2>"v "$2"v <MWE2> PRP @N< 2->3
"<$3>"v "$3"v <MWE3> PROP @P< 3->1)
TARGET ("<(.+?)=(aos?|às?|com|contra|d[eao]s?|em|
n[ao]s?|para)=(.*)>"r PROP /\(@.*\)/r) ;

(1b) SPLITCOHORT:three->fourpart-adv(
"<$1e>"v "de"v <sam-> <MWE1> <MWE:
$1$2=$3=$4>v PRP VSTR:$5 1->p
"<$2>"v "$2"v <-sam> <MWE2> ADV @P< 2->1
"<$3>"v "$3"v <MWE3> PRP VSTR:$5 @P< 3->1
"<$4>"v "$4"v <MWE4> ADV @P< c->3)
TARGET ("<([dD])(ali)=(para)=([^=]+?)>"r ADV \
(@.*\)/r) ;

(1c) SPLITCOHORT (
"<por>" "por" <sam-> <MWEprp> PRP @N< c->p
"<$1>"v "o" <-sam> <artd> <MWEdet> DET M S
@>N 2->p)
        TARGET ("pel([ao]s?)"r)
        (0 PRP OR N/PROP) ;

(1d) SUBSTITUTE (M) (F)
TARGET ("<.*[aà]s?>"r <MWEdet>) ;
```

3.2 Contractions

Fusion of tokens does not add linguistic information, and a function tag can simply be inherited from the head token of the to-be-fused words. Still, CG rules like (2-3) are an effective option for this purpose, too, because the formalism will automatically handle the resulting dependency number adjustments for the rest of the tree, and morphophonetic changes can be addressed where necessary. Here, we use fusion rules to reassemble Portuguese contractions that were split into lemma parts in the original

treebank (marked <sam-> for first part and <-sam> for second part). Thus, (2a) creates a compound POS for the contraction, substituting it for the preposition POS of the contraction's first part. (2b-c) then fuse the tokens "por" and "as" into "pelas", and (2d) creates a compound lemma for the contraction. (3), finally, removes the now-superfluous second part token.

```
(2a) SUBSTITUTE (PRP) (PRP_DET)
        TARGET (<sam->)
        (1 (<-sam> DET)) ;

(2b) SUBSTITUTE
        ("<$1>"v) (VSTR:"<$1$2>")
        TARGET ("<(.*)>"r PRP_DET)
        (1 ("<(.*)>"r <-sam>)) ;

(2c) SUBSTITUTE
        ("<por$1>"v) (VSTR:"<pel$1>")
        TARGET ("<por(.*)>"r PRP_DET) ;

(2d) SUBSTITUTE ("$1"v) (VSTR:"$1+$2")
        TARGET ("([^<]+)"r PRP_DET)
        (1 ("([^<]+)"r <-sam>)) ;

(3) REMCOHORT REPEAT (<-sam>)
        (-1 (/^PRP_.*$/r) OR (PERS_PERS)) ;
```

4 Evaluation and statistics

We evaluated the CG-based retokenization method on the Portuguese Floresta Sintá(c)tica treebank (Afonso et al. 2002), a 239,899 token treebank covering the European as well as the Brazilian varieties of Portuguese. The treebank is available in both constituent and dependency formats, both adhering to the cross-language VISL annotation standard[1]. Since the treebank's native format does not adhere to atomic tokenization, as advocated by the Universal Dependency initiative, retokenization has become an issue for ML-users of the Floresta Sintá(c)tica. Our retokenizer proved capable of addressing this problem, resolving all 8779 MWEs in the treebank into their 21,954 parts (2.50 per MWE), and reestablishing all 15,912 contractions. The process took 33.6 seconds on a 2-core laptop, amounting to a processing speed of 7,140 words/sec.

In combination with a live parser run, on a Portuguese newstext corpus with ~ 1.1 million tokens, the method handled 44,826 MWEs of similar complexity (109,320 parts, 2.44 per MWE), missing out on only 273 (0.6%) MWEs. The failure rate for contractions was a negligible 0.01% (with 76,610 successful fusions).

5 Conclusions and outlook

We have shown that (cg3-level) Constraint Grammar can be used for retokenization, and that our method can establish space-based tokenization for treebanks or parser output that for syntactic or semantic reasons use different tokenization strategies. Thus, both splitting of fused multi-word-expressions and fusion of split contractions can be handled with a high degree of accuracy. In addition to retokenization itself, the method specifically supports tree strucctures in dependency treebanks, preserving ingoing and outgoing dependency arcs and allowing the addition of internal tags and dependency structure for MWEs.

Apart from the treebank conversion discussed here, we hope that the technique will prove useful at various stages of NLP pipelines, supporting grammar- and context-driven tokenization *after* the preprocessing stage, or as post-processing for named entities, numerical expressions or compound-related spelling errors. As a specialized application, we are currently experimenting with target-language retokenization in machine translation.

References

Afonso, Susana & Eckhard Bick & Renato Haber & Diana Santos. 2002. Floresta sintá(c)tica: A treebank for Portuguese. In Proceedings of LREC'2002, Las Palmas. pp. 1698-1703, Paris: ELRA

Bick, Eckhard & Tino Didriksen. 2015. CG-3 - Beyond Classical Constraint Grammar. In: Beáta Megyesi: Proceedings of NODALIDA 2015, May 11-13, 2015, Vilnius, Lithuania. pp. 31-39. Linköping: LiU Electronic Press. ISBN 978-91-7519-098-3

Grefenstette, Gregory & Pasi Tapanainen. 1994. What is a word, what is a sentence? Problems of tokenization. Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX'94), Budapest. pp. 79-87

Kaplan, Ronald M. 2005. A method for tokenizing text. In: Festschrift in Honor of Kimmo

[1] http://visl.sdu.dk

Koskenniemi's 60th anniversary. CSLI Publications, Stanford, CA. pp. 55-64

McDonald, Ryan et al. 2013. Universal dependency annotation for multilingual parsing. In: Proceedings of ACL 2013, Sofia. pp. 92-98