Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 7161–7169
Marseille, 11–16 May 2020

A Tree Extension for CoNLL-RDF

Christian Chiarcos, Luis Glaser
Applied Computational Linguistics Lab (ACoLi)
Goethe University Frankfurt, Germany
{chiarcos, lglaser}@informatik.uni-frankfurt.de

Abstract

The technological bridges between knowledge graphs and natural language processing are of utmost importance for the future development of language technology. CoNLL-RDF is a technology that provides such a bridge for popular one-word-per-line formats with tab-separated values (TSV) that are widely used in NLP (e.g., the CoNLL Shared Tasks), annotation (Universal Dependencies, Unimorph), corpus linguistics (Corpus WorkBench, CWB) and digital lexicography (SketchEngine): Every empty-line separated table (usually a sentence) is parsed into an graph, can be freely manipulated and enriched using standard graph technology, and then be serialized back into in a TSV format, RDF or other formats. An important limitation is that CoNLL-RDF provides native support for word-level annotations only. This does include dependency syntax and semantic role annotations, but neither phrase structures nor text structure. We describe the extension of the CoNLL-RDF technology stack for two vocabulary extensions of CoNLL-TSV, the bracket notation used in earlier CoNLL Shared Tasks and the extension with XML markup elements featured by CWB and SketchEngine. In order to represent the necessary extensions of the CoNLL vocabulary in an adequate fashion, we employ the POWLA vocabulary for representing and navigating in tree structures.

Keywords: linguistic annotation, data formats, interoperability, tab-separated values, RDF, graphs, trees, CoNLL-RDF, POWLA

1. Introduction & Motivation

Since their original conception in the early 1990s, one-word-per-line formats, mostly based on tab-separated values (TSV), have become enormously influential in corpus linguistics (Evert and Hardie, 2011, Corpus Workbench/CWB),[1] digital lexicography (Kilgarriff et al., 2014, SketchEngine),[2] natural language processing (Schmid, 1994, TreeTagger), and as exchange format of various annotation projects (Universal Dependencies (Nivre et al., 2016), PropBank, UniMorph).[3] The main reasons for their continuing popularity are the apparent simplicity of tables and tab-separated values, and their expressive power that allows for the flexible and seamless extension of annotations with additional columns and novel annotations, making these formats capable of expressing and combining any kind of word-level annotation in a compact, human-readable, and machine-processable format.

Yet another factor contributing to the popularity of TSV formats in natural language processing is their application as exchange and evaluation format(s) of the Shared Tasks of the SIGNLL Conference on Computational Natural Language Learning (CoNLL)[4], conducted annually since 1999. With CoNLL data sets as an important basis for future evaluation of most NLP tasks, their formats are supported by most existing NLP tools, and various CoNLL-TSV formats ('dialects') have seen a steady increase in popularity for linguistic data in NLP applications since. Even when created independently from CoNLL Shared Tasks, novel TSV formats following CoNLL conventions are now usually referred to as 'CoNLL-TSV', e.g., the CoNLL-U format of the Universal Dependencies. Here, we follow this convention in describing all one-word-per-line TSV formats as CoNLL-TSV.

Chiarcos and Fäth (2017) introduced the CoNLL-RDF library, which allows to transform *any* CoNLL-TSV format into an isomorphic graph representation in RDF, to flexibly transform it with SPARQL-Update, to enrich it with external ontologies and knowledge bases, and to serialize it in a canonical format, in any RDF serialization (RDF/XML, Turtle, JSON-LD), a graphical representation (using GraphViz/Dot) or back into a CoNLL-TSV representation. This transformation is generic and applicable to any TSV format as it uses user-provided column labels to produce properties (labeled edges) that hold the various annotations. The only data structures it enforces are word (row) and sentence (empty-line separated group of rows), as well as `conll:HEAD`, a property that links a word with its parent (in dependency annotation), resp., the (virtual root of the) sentence. Beyond that, CoNLL-RDF is semantically shallow and its annotation properties are not backed by an ontology. However, using SPARQL Update, CoNLL-RDF data can be flexibly transformed into a proper Linked Data representation and linked with external ontologies.

The canonical CoNLL-RDF format is a fragment of RDF/Turtle that emulates the original TSV formats by putting one word (with its annotations) in one line, and separating sentences with an empty line. However, whereas CoNLL-TSV is a tabular format, the canonical CoNLL-RDF format uses explicit property names (comparable to a Python dictionary or Java hashtable) as its data structure. Aside from providing RDF output, we also provide transformation back into CoNLL for further processing of its annotations with conventional NLP pipelines and libraries.

Despite its popularity and wide use, CoNLL-TSV is limited in its expressivity: It imposes constraints on the type

[*] The authors contributed equally to the submission.
[1] `http://cwb.sourceforge.net/`
[2] `http://www.sketchengine.eu`
[3] `http://universaldependencies.org/`, `http://propbank.github.io/`, `http://unimorph.github.io/`
[4] `http://www.signll.org/conll`

```
#ID WORD      LEMMA    UPOS  POS FEATS        HEAD EDGE
1   James     James    PROPN NNP Number=Sing  2    name
2   Baker     Baker    PROPN NNP Number=Sing  3    nsubj
3   told      tell     VERB  VBD Mood=Ind|... 0    root
4   reporters reporter NOUN  NNS Number=Plur  3    obj
5   Friday    Friday   PROPN NNP Number=Sing  3    nmod:tmod
6   :         :        PUNCT :   _            3    punct
```

Figure 1: Example of CoNLL-U annotation (DEPS and MISC columns omitted)

of linguistic annotation it captures. Indeed, CoNLL-TSV is ideally suited for word-level annotations such as part-of-speech tags or syntactic dependencies, it can be extended to the annotation of non-recursive spans (using the IOBES scheme) and semantic roles (by adding one argument column per predicate), but the format provides no native support for phrase-level annotations, text structure or annotations beyond single sentences. CoNLL-RDF does not impose such restrictions, but allows to create and to process arbitrary graph data structures, for example, for parsing topological fields in Middle High German (Chiarcos et al., 2018), or for the creation of syntactic-semantic annotations in the context of Role and Reference Grammar (Chiarcos and Fäth, 2019). However, as its vocabulary is grounded in CoNLL-TSV, CoNLL-RDF does not specify datatypes for trees and directed graphs not grounded in individual words, nor does the CoNLL-RDF package parse or serialize such annotations.

In this paper we introduce the extensions we made to the CoNLL-RDF library to natively support tree-like annotations. Using the POWLA vocabulary (Chiarcos, 2012c), we model an annotation graph independent of the sentence and word structure imposed by CoNLL. In this way, we can even represent generic linguistic annotations that go beyond token-level annotation e.g. for usage in spoken discourse.

We will first revisit the original concept and functionality of the CoNLL-RDF libraries. Second, this paper will describe a number of one-word-per-line TSV dialects with extensions for tree structures. We show that CoNLL-RDF was only partially able to represent them, and incapable of processing them effectively. We then explain how we complement the CoNLL-RDF vocabulary with generic linguistic data structures from the POWLA vocabulary (Chiarcos, 2012c), and we describe the extensions of the CoNLL-RDF libraries to parse tree annotations, to process and to produce such data structures. Afterwards we present the extensions we made to CoNLL-RDF that enable the tree extension and finish with an outlook on future work.

2. CoNLL-TSV and CoNLL-RDF

2.1. The CoNLL-TSV Format

As the exchange, evaluation and data format for most CoNLL Shared Tasks in the past 20 years, shared task organizers used line wise, tab separated formats with one word per line and sentences split by empty lines. Since then, CoNLL formats have been been used in a variety of applications in NLP beyond the competition. In general, CoNLL-TSV data is represented as tab separated values, each line corresponding to a single word, with each column entry representing an annotation to that word. Sentences are delimited with a new line. There are different CoNLL dialects, that each represent different information in these columns.

Fig. 1 depicts an example representation of such a data format, the CoNLL-U format used in the context of the Universal Dependencies initiative, with the last two columns DEPS and MISC omitted. The column names are given in the first comment line, which is lead by a hash mark. The ID column contains a sentence wide identifier, WORD represents the form, LEMMA the lemma, UPOS the POS-tag in the UD tagset, the POS column contains any language specific speech tag, FEATS contains morphosyntactic features encoded as key-value pairs with a pipe ('|') as delimiter. The HEAD column points to the ID of the parent in the dependency tree, with 0 representing the root. Finally, the EDGE column carries the label of the dependency relation that holds between a word and its head, resp., the (virtual root of the) sentence.[5]

CoNLL-U only represents one specific CoNLL dialect, but is supported by most multilingual dependency parsers – as these are usually trained on UD data. Fig. 1 shows a slightly simplified example from the OntoNotes (Hovy et al., 2006) `wsj-0655` file, with annotations subsequently provided by the authors. Other dialects omit, reorder or complement these columns with additional content. All of them share the general advantages of this format: They enjoy widespread integration in existing NLP technology, they are easy to read and can be transformed and read without a lot of overhead.

2.2. From TSV to Annotation Graphs

Being easily processable, interpretable and extensible, tabular data structures in TSV formats are popular and effective for word-level annotations. They are also capable of representing other annotations, e.g., dependency relations between words by means of coindexing (between HEAD and ID columns in Fig. 1). But while TSV formats can be trivially parsed into tables (two-dimensional arrays), such tables are less than ideal for querying or manipulating graph structures. With the following listing, we show how to retrieve the lemmas of all subject arguments of the root nodes of a sentence in Java, using a two-dimensional array:

```
String[][] sentence; // 2-dimensional array
// find root
for(int i = 0; i<sentence.length; i++)
 if(sentence[i][7].equals("root")) {
  String[] root = sentence[i];
  // find nsubj
  for(int j = 0; j<sentence.length; i++)
   if(sentence[j][6].equals(root[0]) &&
      sentence[j][7].equals("nsubj")) {
       String[] nsubj = sentence[j];
       // result value: lemma
       String lemma = nsubj[2]; } }
```

[5] CoNLL formats differ with respect to the order and content of columns, but also with respect to the labels they apply. In CoNLL-U terminology, the columns WORD and EDGE are abbreviated FORM and DEPREL, respectively. Here, we stay with CoNLL-RDF conventions.

The example nicely illustrates that path traversal within array requires loops and variable bindings for every edge in the graph. In particular, it is hardly possible to define graph patterns of variable length. Graphs provide the higher-order data structures necessary for this purpose; with RDF and SPARQL, they can be represented, queried and manipulated in a platform-independent, W3C-standardized manner. Another advantage of using RDF for representing graphs is that it builds on Uniform Resource Identifiers (URIs) to represent nodes, relations and types. With a URI, every node *requires* the developer to provide a globally unambiguous identity, whereas CoNLL-U IDs are only unique within a sentence. As a result, RDF trivially permits creating cross-references between different sentences, whereas CoNLL-U does not provide a formalism for this purpose, so that ad hoc solutions need to be improvised.[6] CoNLL-RDF was designed to provide a seamless round-tripping between arbitrary CoNLL-TSV formats and RDF graphs.

2.3. CoNLL-RDF

Before we proceed with the extension of CoNLL-RDF, we describe its current status. CoNLL-RDF (Chiarcos and Fäth, 2017) is available as open source from our GitHub repository.[7] CoNLL-RDF provides vocabulary conventions for representing CoNLL and other TSV formats in RDF, components for consuming and producing such data, a component for the efficient manipulation of CoNLL data by means of SPARQL updates applied in parallel to individual sentence graphs, and various scripts that demonstrate the potential of this functionality.

Using the CoNLL-RDF library, linguistic data in any CoNLL dialect or TSV format can be transformed to a shallow RDF representation of it. CoNLL-RDF builds on a small fragment of the popular NIF vocabulary (Hellmann et al., 2013), i.e., the concepts `nif:Sentence`, `nif:Words` and the properties `nif:nextSentence` and `nif:nextWord`.[8] Beyond this, CoNLL-RDF adopts user-provided column labels as names of properties in the conll name space. It is thus a freely extensible, but a semantically shallow vocabulary that is not being backed up by a formal ontology. All column labels are preserved as data type properties that provide string values, except for very few that carry special semantics: HEAD will be converted to an object property (foreign key) pointing from the current word to its syntactic head, ID (if present) is used to define the word URI, and PRED_ARGs is a label that captures multiple columns for semantic role annotation.[9]

```
:s1_1 rdf:type      nif:Word;
      conll:ID      "1";
      conll:WORD    "James";
      conll:LEMMA   "James";
      conll:UPOS    "PROPN";
      conll:POS     "NNP";
      conll:FEATS   "Number=Sing";
      conll:HEAD    :s1_2;
      conll:EDGE    "name";
      nif:nextWord  :s1_2 .
```

Figure 2: CoNLL-RDF transformation of the first row in fig. 1

The conversion with CoNLL-RDF returns a canonical serialization in Turtle/RDF of the CoNLL data using user-specified column names in the conll namespace. Thus, the new representation can be further enhanced with SPARQL update queries using the CoNLLRDFUpdater. Because we emphasize interoparability with existing NLP and LOD technology, we also provide the CoNLLRDFFormatter that can recreate CoNLL from the CoNLL-RDF after updates have been performed. Note that the last step may be lossy, as newly introduced annotations may not be representable within the restrictions of CoNLL.

2.3.1. The CoNLL-RDF Format

The RDF data model is grounded on the notion of triples, consisting of a subject (source node), property (relation type) and object (target node or literal value), each represented by a URI, say, `<http://ufal.mff.cuni.cz/conll2009-st/task-description.html#ID>` for the ID column. The Turtle format allows to define namespace prefixes (e.g., `conll:ID`) and separates full triples by a dot (.). If a triple shares its subject with the preceding triple, it is possible to omit the subject and to mark this by using semicolon (;) as triple separator. The *canonical* CoNLL-RDF format is a Turtle fragment that requires sentence breaks to be marked by empty lines, and for every word a word URI followed by a list of semicolon-separated properties (conll properties, `nif:nextWord`, other RDF properties). Information on the sentence precedes or follows all words of the sentence, again written in a single line.

Fig. 2 provides a fragment of CoNLL-RDF in its Turtle serialization. In the canonical CoNLL-RDF format, this information would be written in a single line, adjusted here for presentational reasons.

2.3.2. Querying CoNLL-RDF

The dominant query language for RDF data is SPARQL (Prud'Hommeaux and Seaborne, 2008), a W3C standard that can be informally described as a combination of aspects of SQL (SELECT, WHERE, BIND, FIL-

[6] The CoNLL-2011 format added a COREF column that provides co-indexing information. The CoNLL-2015 format introduced an additional column with numerical word IDs defined on text level rather than sentence level. We are not aware of a CoNLL-TSV solution to express relations across different texts.

[7] `https://github.com/acoli-repo/conll-rdf`, Apache license 2.0.

[8] CoNLL-RDF deviates from NIF in providing its own URI schema. This is necessary, because offset-based string URIs as foreseen in NIF are not applicable to a format that does not preserve the original white spaces after tokenization.

[9] Since CoNLL-2004, semantic role information is represented by multiple columns: For every predicate in the PRED column, another argument row is introduced that holds the argument information for this particular predicate. In CoNLL-RDF, this is represented by object properties pointing from the predicate (word) to the argument (words), and which take their name from the annotated semantic role (e.g., `conll:A0` for the agens argument, usually annotated as A0).

TER, etc.) and Turtle (representation of graph patterns, extended with variables). In addition, SPARQL 1.1 (Harris and Seaborne, 2013) introduced property paths, i.e., the possibility to express chains of triples (e.g., `?x conll:HEAD/conll:HEAD ?z` as a shorthand for `?x conll:HEAD ?y. ?y conll:HEAD ?z`), and operators over these, e.g., the Kleene star (`?x conll:HEAD* ?y`).
The lemma(s) of the subject(s) of root nodes in CoNLL-U can thus be addressed in a SPARQL WHERE block:

```
?root  conll:EDGE "root".
?nsubj conll:EDGE "nsubj".
?nsubj conll:HEAD ?root.
?nsubj conll:LEMMA ?lemma.
```

2.3.3. Manipulating CoNLL-RDF

The CoNLL-RDF library provides the CoNLLRDFUpdater that allows to apply and to iterate a sequence of SPARQL Update scripts to the CoNLL data it consumes. This can be used, for example, to simplify navigation in a parse tree. The following SPARQL Update script combines information from EDGE and HEAD columns to facilitate navigation in CoNLL-U dependency trees:

```
INSERT { ?x ?rel ?y.}
WHERE {
 ?x conll:HEAD ?y; conll:EDGE ?e.
 BIND(URI(CONCAT('https://┘
 ↪  universaldependencies.org/u/dep/',?e))
 ↪  AS ?rel) }
```

As a result, we can now effectively traverse UD dependencies:[10]

```
PREFIX dep: <https://┘
↪  universaldependencies.org/u/dep/>
SELECT ?lemma
WHERE {
 ?nsubj dep:nsubj/dep:root ?sentence;
        conll:LEMMA ?lemma   }
```

More complex graph patterns can be expressed using additional SPARQL property path operators such as + and * (iteration), | (disjunction), ^ (inversion), resp., their corresponding grouping in parentheses.
As mentioned above, CoNLL-RDF has been used for various annotation and feature extraction tasks. It does however, inherit some of the limitations of the CoNLL-TSV data structures it is based on: CoNLL-TSV can neither adequately represent units of annotation that are smaller than words (e.g., in morphology), nor is it capable of representing nested structures (as necessary for text structure or phrase structure grammar). Each of these can be represented in RDF, but they require to extend the CoNLL-RDF vocabulary beyond elements defined by CoNLL, and these vocabulary elements need to be supported by the CoNLL-RDF package as input and output elements.

[10]Note that we also improved annotation transparency: The URIs of the newly created properties are links to the actual documentation about the corresponding label, so, this information is now bundled with the annotation graph.

In the following section, we describe two extensions of the original one-word-per-line TSV formats introduced for representing tree annotations, and use these to extend the CoNLL-RDF infrastructure accordingly.

3. Tree Structures in TSV Formats

Conversion from CoNLL to CoNLL-RDF works seamlessly for word-level annotations, dependency syntax and semantic role annotations. However, one-word-per-line TSV formats provide rudimentary support for annotations on the level of text (markup) and syntax (phrase structures) only, and CoNLL-RDF inherits this limitation.
We present two suggested extensions of one-word-per-line TSV formats as the basis of the extension of CoNLL-RDF with data structures for trees: the bracketing notation in accordance with the Penn Treebank and the XML markup used by SketchEngine or CorpusWorkbench.

3.1. PTB Bracketing Notation

The Penn Treebank(Marcus et al., 1993, PTB) featured a formalism for phrase structure grammar that uses opening and closing brackets to indicate beginning and end of a phrase. They enclose first the phrase annotation followed by the primary data. For representing such data, the CoNLL-2005 format added another column with parse information split into word-level pieces, and the primary data (= content of the WORD column) replaced by the placeholder '*'. For each unit of primary data, the bracketing and annotations containing the *-replacement are presented in a separate column.
In CoNLL-RDF, this information is simply preserved as an opaque string, e.g., in the triple `:s1_1 conll:PARSE "(TOP (S (NP-SBJ *"` for the first row in Fig. 4.
With the CoNLLRDFUpdater, it was already possible to recover the actual tree structure from such strings using SPARQL Update scripts, and likewise, to create them from an internal data structure that represents the parse tree. This is, however, too complex and time-consuming to be practical. In particular, it requires recursive SPARQL Updates in order to find and resolve (resp., insert) matching parentheses. As an alternative, we provide native support for tree structures in CoNLL-RDF.

3.2. SGML/XML Markup Extentions

For representing markup information and syntactic chunks, the CorpusWorkbench (Evert and Hardie, 2011) and the SketchEngine (Kilgarriff et al., 2014) provide another extension of one-word-per-line TSV formats, the enrichment with XML (SGML) markup elements: In order to represent trees, word-level annotations can be interrupted by lines that hold one markup element each. For example, several lines of word-level annotations can be grouped together into a phrase by enclosing them between single lines with the markup elements `<p>`, resp. `</p>`. XML markup may also include document-wide information, for which the CoNLL-TSV formalism has no concept. See Fig. 6 for example data from SketchEngine.
In the original CoNLL-RDF pipeline, this markup information was largely ignored, except that certain markup

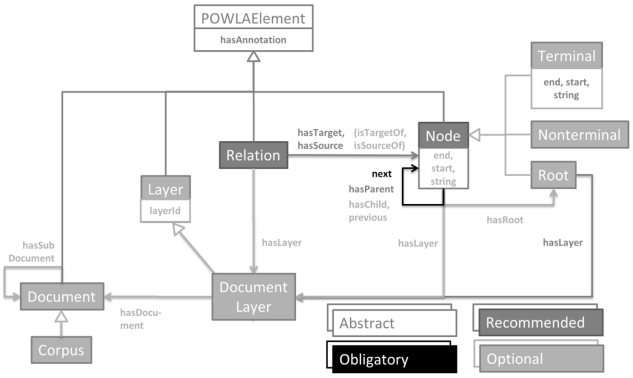

Figure 3: POWLA data model

elements were recognized as sentence separators (like an empty line in CoNLL-TSV).

4. Extending the CoNLL-RDF Vocabulary with POWLA Data Types

CoNLL-TSV data structures provide no base vocabulary for representing trees. As we aim to use established vocabularies rather than to invent new ones, we adopt the POWLA vocabulary (Chiarcos, 2012b), as this provides an OWL2/DL implementation of the Linguistic Annotation Framework (Ide and Romary, 2004, LAF). In this approach, we follow the suggestion of Cimiano et al. (2020, p.103-111) to complement NIF with POWLA data structures and extend it to CoNLL-RDF.

4.1. The POWLA Vocabulary

POWLA (Chiarcos, 2012c) is an OWL2DL vocabulary for systematizing linguistic annotations. Compared to other formalisms, POWLA does not make any assumptions on what the atomic structures look like. POWLA is an OWL2/DL serialization of PAULA (Dipper and Götze, 2005; Dipper and Götze, 2006)[11] an early implementation of LAF (Ide and Romary, 2004) and serialized in a standoff XML format (Stede et al., 2006; Zeldes et al., 2009). POWLA has been applied to modeling multi-layer corpora, including case studies on the Manually Annotated Sub-Corpus of the American National Corpus (Ide et al., 2008, MASC), syntactic and coreference annotated corpora (Chiarcos, 2012a), high-precision information extraction (de Araujo et al., 2017), annotation engineering (Chiarcos and Fäth, 2019), and syntactic parsing (Chiarcos et al., 2018).

POWLA aims to formalize linguistic annotations by building on existing standards with respect to their anchoring in the original document. PAULA XML uses XLink/XPointer references for this purpose. POWLA's design goal is to complement existing NIF, Web Annotations or application-specific RDF renderings of linguistic annotations with interoperable linguistic data structures. It is thus underspecified in respect to the anchor modelling.

[11] Every PAULA data structure can be expressed in POWLA, but POWLA is slightly more general: It is *not* restricted to acyclic graphs. This is not a disadvantage, as cycle detection (and resolution) for POWLA data can be easily implemented by SPARQL, e.g., using the property path `powla:hasParent`.

```
James     NNP     (TOP (S (NP–SBJ *
Baker     NNP     *)
told      VBD     (VP *
reporters         NNS     (NP * )
Friday    NNP     (NP–TMP * )
:         :       *
```

Figure 4: Example of CoNLL bracketing notation

```
James     NNP     (TOP (S (NP–SBJ *
Baker     NNP     * )
told      VBD     (VP *
reporters         NNS     (NP * )
Friday    NNP     (NP–TMP * )
:         :       * )))
```

Figure 5: CoNLL with bracketing notation after round-tripping through CoNLL-RDF, identical with Fig. 4 except for whitespaces

4.2. Key Concepts of POWLA

The overall structure of the POWLA vocabulary is summarized in Fig. 3. POWLA defines a key concept in form of `powla:Node`. This node expresses a single linguistic annotation. These nodes can be related to each other using two property pairs. The property `powla:hasParent` (`powla:hasChild` is its inverse) can be used to express hierarchical structures in that annotation, e.g. when representing syntax trees. The property `powla:next` (`powla:previous` is its inverse) can be used to put `powla:Nodes` into a sequential order. We recommend to always put siblings pointing to the same node with `powla:hasParent` in relation with each other by using `powla:next`, as this makes navigating the graph easier. Also, this provides an order of the tree elements that can be independent of the underlying strings. This can be useful when annotating speech data for example, where the canonical syntax may deviate from the actual representation.

5. Processing Tree Annotations

5.1. Penn Treebank Bracketing Notation

We now elaborate on how this transformation works for CoNLL containing PTB-like bracketing annotation as described in section 3.1. First, we detect columns that contain bracketing annotation by searching for open and closing bracket, because we need to treat them separately from the normal CoNLL columns. Then, we read the CoNLL-TSV sentence wise, recognizing new lines as sentence boundaries. For each sentence, we first separate standard CoNLL-TSV columns, treat them as described in section 2.1. and output them. With the remaining columns containing bracketing annotation we do the following: We remove the aforementioned *-placeholders in all columns that contain the bracketing annotations. We then traverse the tree given by the bracketing annotation within

```
<doc id="G10" n="32">
<head type="min">
FEDERAL JJ
CONSTITUTION NN
<g/>
, $,
1789 NUM
</head>
<p n="1">
" $,
<g/>
we PRP
the DT
People NN
```

Figure 6: Example of a SketchEngine file

the POWLA structure. Similar to the `conll:COLNAME` structure we employ when parsing CoNLL, we introduce a `powla:Node` that also has a `conll:COLNAME` object for each annotation we find. This node will furthermore be linked to other `powla:Node`s via `powla:hasParent` and `powla:next` properties to recreate the tree structure. The annotation part of the tree will be represented with a `rdf:value` property. Fig. 4 displays a partial sentence in such a CoNLL file with bracketing notation. Using the CoNLLBrackets2RDF class, we generate the shallow CoNLL-RDF transformation depicted in Fig. 8.

Because one central design goal of the CoNLL-RDF package is building a bridge between NLP and LOD formats and pipelines, we also provide functionality for round-tripping to CoNLL again. Using a SPARQL update query, we again collapse the tree structure that the `powla:Node`s represent into their former bracketing representation. This is achieved by using the link established between the properties in the `conll:` namespace and the `powla:Node` structure. For each `conll:` property, we search the graph linked via `powla:hasParent` for all dependent `powla:Node`s and concatenate their `rdf:value`s into the original bracketed notation and reintroduce the *-placeholder. The result after roundtripping is depicted in Fig. 5. Also note, that the brackets were incomplete in Fig. 4, because we only used a partial sentence. This is fixed by the SPARQL script, as we guarantee full trees.

5.2. SGML/XML Extensions

In analogy with section 5.2., we provide a conversion of the SGML/XML markup extensions to CoNLL-RDF. XML data that is captured, will be appended to a `powla:Node` with type `conll:XML_DATA`. The node name will be given as an object to a `rdf:value` property. Each attribute contained in the XML node will receive a dedicated property in the xml namespace[12] with the attribute value as the object. E.g. the `doc` XML node in the first line in Fig. 6 will result in `:x1 a powla:Node, conll:XML_DATA; rdf:value "doc" ; x:id "G10"; x:n "32".`. See Fig. 9 for the full CoNLL-RDF result. Again, as we emphasize

[12] `http://purl.org/acoli/conll-rdf/xml#`

```
# WORD  POS     XML_DATA
FEDERAL  JJ     (doc n=\"32\" id=\"G10\" (head type=\"min\" *
CONSTITUTION    NN      *
        -       (g * )
,       $,      *
1789    NUM     * ))

&quot;  $,      (doc n=\"32\" id=\"G10\" (p n=\"1\" *
        -       (g * )
we      PRP     *
the     DT      *
People  NN      * ))
```

Figure 7: CoNLL with XML data in PTB-style annotation after roundtrip

interoperability between NLP and LOD in our design, we provide functionality for reconversion to CoNLL. Note that we can not fully reconstruct the XML annotations, as they may give information that is valid for the entire document which is not possible in the standard CoNLL format. Instead, we can recover the tree structure similar to the PTB-like annotations as explained in section 5.1. which results in CoNLL as depicted in Fig. 7. Parsing the example SketchEngine file in Fig. 6 with the generic CoNLLStreamExtractor class yielded 41 triples compared to 72 triples using the new XMLTSV2RDF class. During further postprocessing scripts, the original XML markup can be restored.

5.3. Limitations

SketchEngine allows non-closing brackets, which we currently do not. Therefore it must be ensured that each opening xml span will be closed at some point. Furthermore, we do not validate the result in any way. To detect and resolve cycles, we provide a SPARQL script at our GitHub repository. Alternatively, it is possible to use dedicated solutions for this purpose, for example Shape Expressions (ShEx).[13]

6. Evaluation

We conducted three evaluation experiments:

1. Runtime for parsing CoNLL-TSV with PTB-style syntax annotations
2. Information loss for round-tripping from CoNLL-TSV via CoNLL-RDF and POWLA to CoNLL-TSV
3. Conversion from SketchEngine format via CoNLL-RDF and POWLA to CoNLL-TSV

For experiment 1, we compare an implementation that uses SPARQL Update scripts with the revised CoNLL-RDF parser. We use both to convert the WSJ section of the OntoNotes corpus,[14] to CoNLL-RDF using a 3.79 GHz Intel i5-7600K with 16 GB of memory and a SSD drive. Table 1 presents a significant speed-up.

For experiment 2, we parsed and regenerated the WSJ section of the OntoNotes corpus. For this purpose, we excluded the SRL annotations from the documents. We

[13] `http://shex.io/`

[14] PropBank edition, available from `https://github.com/propbank/propbank-release`

Version	number of triples	time elapsed	time per triple
(1)	10,924,378	966,162 ms	0.088 ms
(2)	12,526,204	34,457 ms	0.0027 ms

Table 1: Timing comparisons between conversion with (1) CoNLLStreamExtractor + SPARQL updates and (2) CoNLLBrackets2RDF

were able to perform a loss-less roundtrip from the original CoNLL-TSV to a CoNLL-RDF and POWLA representation and back to an identical representation of the data in CoNLL-TSV. Parsing of SRL arguments was already possible using the CoNLLStreamExtractor.

For experiment 3, the principal feasibility has been demonstrated in section 3.2.. Because we only recover a PTB-like CoNLL annotation of the XML-markup elements so far, we did not perform a round-tripping experiment as in experiment 2. Instead, we measured whether the information contained in the XML markup is adequately preserved by comparing the number of markup elements with the number of `powla:Nodes` retrieved from it. We transformed part of the TenTen Corpus (Jakubíček et al., 2013) into CoNLL-RDF using the SGML/XML extension. The SGML/XML annotations are embedded in metadata concerning the retrieval, which we removed. This resulted in 71,617 `powla:Nodes` representing the annotation graph. This is identical to the number of XML markup elements. This means, that we were able to fully transport all SGML/XML annotations from the original corpus into CoNLL-RDF.

7. Related Research

A very large number of interchange formats for NLP and representation formalisms for linguistic annotations have been (and continues to be) developed in the past 10 years, some of them developed as native RDF vocabularies, usually with serializations in Turtle or JSON-LD, many others with designated interfaces with Semantic Web technology, e.g., NAF (Fokkens et al., 2014).[15]

Popular representatives from the first group are the NLP Interchange Format (Hellmann et al., 2013, NIF), and the LAPPS Interchange Format (Verhagen et al., 2015, LIF). Both provide basic data structures for linguistic annotations in NLP pipelines, coupled with annotation-specific extensions. NIF focuses on formalizing string annotations (and is employed here for this purpose). It does not, however, provide specifications for linguistic data structures *in general*. In particular, NIF 2.0 provides no data structures for empty nodes (e.g., syntactic traces or zero anaphora), nor does it permit multiple distinct, but co-extensional annotations of the same string. Although it does provide syntax-related classes such as `nif:Phrase` ($\sqsubseteq$ `nif:String`), it is thus not an adequate vocabulary for extending CoNLL-RDF with generic data structures for representing trees.

LIF[16] is a JSON-LD-based formalism inspired by NIF, but capable to express such information. LIF is designed for NLP pipelines and provides data structures for popular types of linguistic annotation. As far as generic data structures for linguistic annotation are concerned, these are largely equivalent to POWLA, although less compact, in that LIF relations are reified, whereas POWLA provides the `powla:hasParent` relation as a single statement.

Another RDF-based vocabulary that has been used for linguistic annotation is Web Annotation (Sanderson et al., 2017, WA). Web Annotation was not designed for NLP or language resources, but for expressing metadata over web objects, and it is largely applied for this purpose, especially in Bioinformatics and Digital Humanities. It does not, however, provide specifically linguistic data structures.

Overall, these formats are actively used by their respective communities, but they are less popular in the language resource community. The reason is pragmatic rather than scientific, i.e., that they impose some technical overhead over working with traditional formats such as CoNLL-TSV, so, for applications not requiring Semantic Web technology (even if they could benefit from it), they tend to be ignored. By providing a technological bridge between RDF and TSV formats, CoNLL-RDF addresses this particular gap and to the best of our knowledge, it is the only proposal of its kind for TSV formats in language technology. Interfaces between TSV formats and RDF technology have been explored before, e.g., with the W3C recommentation CSV2RDF (Tandy et al., 2015), that allows to map columns in a table to RDF properties. It however misses certain requirements for language resources, e.g., support for tables of varying size (for SRL annotations), sentence splitting with empty lines, or relations between adjacent rows (`nif:nextWord`). We thus chose CoNLL-RDF as the basis for our experiments rather than CSV2RDF.

The tree extension of CoNLL-RDF builds on the POWLA vocabulary. POWLA has originally been developed for modelling and querying multi-layer corpora (Chiarcos, 2012b), and subsequently applied in information extraction (de Araujo et al., 2017). Alternatives to POWLA do exist (especially the LIF format mentioned above), and we chose POWLA because it is a compact, generic vocabulary (no annotation-specific extensions, no obligatory reification of hasParent relations).

8. Conclusion & Outlook

In this paper, we present the extension to the CoNLL-RDF library. We present a novel approach to natively transform tree-like structures in two different varieties: PTB-like annotations and CoNLL embedded in XML markup as used by SketchEngine or CorpusWorkbench. To do so, we extend the previous shallow transformation from CoNLL to CoNLL-RDF with a parallel annotation graph using the POWLA vocabulary. This enables us to represent annotations on any level of complexity, even going beyond tree-like structures by representing entire annotation graphs using RDF. The extensions to the CoNLL-RDF libraries enable lossless round-tripping from CoNLL to CoNLL-RDF and back for PTB-like annotations. Round-tripping CoNLL embedded in XML may be lossy as document-wide annotations cannot be represented in standard CoNLL. Instead we provide conversion from XML embedded CoNLL to CoNLL-RDF with a possible roundtrip to a PTB-like representation in CoNLL.

[15] `https://github.com/newsreader/NAF`

[16] `http://vocab.lappsgrid.org/`

```
PREFIX nif: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>
PREFIX conll: <http://ufal.mff.cuni.cz/conll2009-st/task-description.html#>
PREFIX x: <http://purl.org/acoli/conll-rdf/xml#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX terms: <http://purl.org/acoli/open-ie/>
PREFIX powla: <http://purl.org/powla/powla.owl#>
PREFIX : <http://example.com/>
:s1_0 a nif:Sentence.
:s1_1 a nif:Word; conll:WORD "James"; conll:POS_PTB "NNP"; conll:HEAD :s1_0; nif:nextWord :s1_2.
:s1_2 a nif:Word; conll:WORD "Baker"; conll:POS_PTB "NNP"; conll:HEAD :s1_0; nif:nextWord :s1_3.
:s1_3 a nif:Word; conll:WORD "told"; conll:POS_PTB "VBD"; conll:HEAD :s1_0; nif:nextWord :s1_4.
:s1_4 a nif:Word; conll:WORD "reporters"; conll:POS_PTB "NNS"; conll:HEAD :s1_0; nif:nextWord :s1_5.
:s1_5 a nif:Word; conll:WORD "Friday"; conll:POS_PTB "NNP"; conll:HEAD :s1_0; nif:nextWord :s1_6.
:s1_6 a nif:Word; conll:WORD ":"; conll:POS_PTB ":"; conll:HEAD :s1_0.
:bPARSE_PTB_1 a powla:Node, conll:PARSE_PTB ; rdf:value "TOP" .
:bPARSE_PTB_2 a powla:Node, conll:PARSE_PTB ; powla:hasParent :bPARSE_PTB_1 ; rdf:value "S" .
:bPARSE_PTB_3 a powla:Node, conll:PARSE_PTB ; powla:hasParent :bPARSE_PTB_2 ; rdf:value "NP-SBJ" .
:s1_1 powla:hasParent :bPARSE_PTB_3 .
:s1_1 powla:next :s1_2.
:s1_2 powla:hasParent :bPARSE_PTB_3 .
:bPARSE_PTB_3 powla:next :bPARSE_PTB_4.
:bPARSE_PTB_4 a powla:Node, conll:PARSE_PTB ; powla:hasParent :bPARSE_PTB_2 ; rdf:value "VP" .
:s1_3 powla:hasParent :bPARSE_PTB_4 .
:s1_3 powla:next :bPARSE_PTB_5.
:bPARSE_PTB_5 a powla:Node, conll:PARSE_PTB ; powla:hasParent :bPARSE_PTB_4 ; rdf:value "NP" .
:s1_4 powla:hasParent :bPARSE_PTB_5 .
:bPARSE_PTB_5 powla:next :bPARSE_PTB_6.
:bPARSE_PTB_6 a powla:Node, conll:PARSE_PTB ; powla:hasParent :bPARSE_PTB_4 ; rdf:value "NP-TMP" .
:s1_5 powla:hasParent :bPARSE_PTB_6 .
:bPARSE_PTB_6 powla:next :s1_6.
:s1_6 powla:hasParent :bPARSE_PTB_4 .
```

Figure 8: Shallow CoNLL-RDF transformation of bracketing notation in fig. 4

```
:s1_0 a nif:Sentence.
:s1_1 a nif:Word; conll:WORD "FEDERAL"; conll:POS "JJ"; conll:HEAD :s1_0; nif:nextWord :s1_2.
:s1_2 a nif:Word; conll:WORD "CONSTITUTION"; conll:POS "NN"; conll:HEAD :s1_0; nif:nextWord :s1_3.
:s1_3 a nif:Word; conll:WORD ","; conll:POS "$,"; conll:HEAD :s1_0; nif:nextWord :s1_4.
:s1_4 a nif:Word; conll:WORD "1789"; conll:POS "NUM"; conll:HEAD :s1_0.
:x1 a powla:Node, conll:XML_DATA; rdf:value "doc" ; x:id "G10"; x:n "32".
:x2 powla:hasParent :x1; a powla:Node, conll:XML_DATA; rdf:value "head" ; x:type "min".
:s1_1 powla:hasParent :x2 .
:s1_1 powla:next :s1_2 .
:s1_2 powla:hasParent :x2 .
:s1_2 powla:next :x3. :x3 powla:hasParent :x2; a powla:Node, conll:XML_DATA; rdf:value "g" .
:x3 powla:next :s1_3 .
:s1_3 powla:hasParent :x2 .
:s1_3 powla:next :s1_4 .
:s1_4 powla:hasParent :x2 .

:s1_0 nif:nextSentence :s2_0.
:s2_0 a nif:Sentence.
:s2_1 a nif:Word; conll:WORD "&quot;"; conll:POS "$,"; conll:HEAD :s2_0; nif:nextWord :s2_2.
:s2_2 a nif:Word; conll:WORD "we"; conll:POS "PRP"; conll:HEAD :s2_0; nif:nextWord :s2_3.
:s2_3 a nif:Word; conll:WORD "the"; conll:POS "DT"; conll:HEAD :s2_0; nif:nextWord :s2_4.
:s2_4 a nif:Word; conll:WORD "People"; conll:POS "NN"; conll:HEAD :s2_0.
:x1 a powla:Node, conll:XML_DATA; rdf:value "doc" ; x:id "G10"; x:n "32".
:x4 powla:hasParent :x1; a powla:Node, conll:XML_DATA; rdf:value "p" ; x:n "1".
:s2_1 powla:hasParent :x4 .
:s2_1 powla:next :x5. :x5 powla:hasParent :x4; a powla:Node, conll:XML_DATA; rdf:value "g" .
:x5 powla:next :s2_2 .
:s2_2 powla:hasParent :x4 .
:s2_2 powla:next :s2_3 .
:s2_3 powla:hasParent :x4 .
:s2_3 powla:next :s2_4 .
:s2_4 powla:hasParent :x4 .
```

Figure 9: CoNLL-RDF conversion of the SketchEngine sample presented in fig. 6

9. Acknowledgements

The research described in this paper has been partially conducted in the context of the BMBF Early Career Research Group 'Linked Open Dictionaries (LiODi)', and partially in the context of the Horizon 2020 Research and Innovation Action 'Pret-a-LLOD', Grant Agreement number 825182. We would like to thank Mari Aigro, University of Tartu, for providing us with sample data from the TenTen Corpus for SketchEngine evaluation.

10. Bibliographical References

Chiarcos, C. and Fäth, C. (2017). CoNLL-RDF: Linked corpora done in an NLP-friendly way. In Jorge Gracia, et al., editors, *Language, Data, and Knowledge*, pages 74–88, Cham, Switzerland. Springer.

Chiarcos, C. and Fäth, C. (2019). Graph-based annotation engineering: Towards a gold corpus for Role and Reference Grammar. In *2nd Conference on Language, Data and Knowledge (LDK-2019)*. OpenAccess Series in Informatics, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany.

Chiarcos, C., Kosmehl, B., Fäth, C., and Sukhareva, M. (2018). Analyzing Middle High German syntax with RDF and SPARQL. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan.

Chiarcos, C. (2012a). POWLA: Modeling linguistic corpora in OWL/DL. In *9th Extended Semantic Web Conference (ESWC-2012)*, pages 225–239, Heraklion, Crete, May.

Chiarcos, C. (2012b). A generic formalism to represent linguistic corpora in RDF and OWL/DL. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC-2012)*, pages 3205–3212. European Language Resources Association (ELRA).

Chiarcos, C. (2012c). Powla: Modeling linguistic corpora in owl/dl. In Elena Simperl, et al., editors, *The Semantic Web: Research and Applications*, pages 225–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cimiano, P., Chiarcos, C., McCrae, J., and Gracia, J. (2020). *Linguistic Linked Data. Representation, Generation and Applications*. Springer, Cham.

de Araujo, D. A., Rigo, S. J., and Barbosa, J. L. V. (2017). Ontology-based information extraction for juridical events with case studies in Brazilian legal realm. *Artificial Intelligence and Law*, 25(4):379–396.

Dipper, S. and Götze, M. (2005). Accessing heterogeneous linguistic data — generic XML-based representation and flexible visualization. In *2nd Language & Technology Conference 2005*, pages 23–30, Poznan, Poland, April.

Dipper, S. and Götze, M. (2006). ANNIS: Complex Multilevel Annotations in a Linguistic Database. In *5th Workshop on NLP and XML (NLPXML-2006): Multi-Dimensional Markup in Natural Language Processing*, Trento, Italy.

Evert, S. and Hardie, A. (2011). Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium. In *Proceedings of the Corpus Linguistics 2011 conference*, Birmingham, UK.

Fokkens, A., Soroa, A., Beloki, Z., Ockeloen, N., Rigau, G., van Hage, W. R., and Vossen, P. (2014). NAF and GAF: Linking linguistic annotations. In *Proc. 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*, pages 9–16.

Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. W3C recommendation, World Wide Web Consortium.

Hellmann, S., Lehmann, J., Auer, S., and Brümmer, M. (2013). Integrating NLP Using Linked Data. In Camille Salinesi, et al., editors, *Advanced Information Systems Engineering*, volume 7908, pages 98–113. Springer Berlin Heidelberg.

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). OntoNotes: The 90% solution. In *Proc. of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short 2006, pages 57–60, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ide, N. and Romary, L. (2004). International standard for a linguistic annotation framework. *Natural language engineering*, 10(3-4):211–225.

Ide, N., Baker, C. F., Fellbaum, C., Fillmore, C. J., and Passonneau, R. (2008). MASC: The Manually Annotated Sub-Corpus of American English. In *6th International Conference on Language Resources and Evaluation (LREC-2008)*, pages 2455–2461, Marrakech, Morocco, May.

Jakubíček, M., Kilgarriff, A., Kovář, V., Rychlý, P., and Suchomel, V. (2013). The tenten corpus family. In *7th International Corpus Linguistics Conference CL-2013*, pages 125–127, Lancaster.

Kilgarriff, A., Baisa, V., Bušta, J., Jakubíček, M., Kovář, V., Michelfeit, J., Rychlý, P., and Suchomel, V. (2014). The Sketch Engine: ten years on. *Lexicography*, 1(1):7–36.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguist.*, 19(2):313–330.

Nivre, J., Agić, Ž., Ahrenberg, L., and et. al. (2016). Universal dependencies 1.4. `http://hdl.handle.net/11234/1-1827`.

Prud'Hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. *W3C working draft*, 4(January).

Sanderson, R., Ciccarese, P., and Young, B. (2017). Web Annotation Data Model. Technical report, W3C Recommendation.

Schmid, H. (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK.

Stede, M., Bieler, H., Dipper, S., and Suriyawongk, A. (2006). Summar: Combining linguistics and statistics for text summarization. In *17th European Conference on Artificial Intelligence (ECAI-2006)*, pages 827–828, Riva del Garda, Italy.

Tandy, J., Herman, I., and Kellogg, G. (2015). Generating RDF from tabular data on the web. Technical report, W3C Recommendation.

Verhagen, M., Suderman, K., Wang, D., Ide, N., Shi, C., Wright, J., and Pustejovsky, J. (2015). The LAPPS Interchange Format. In *International Workshop on Worldwide Language Service Infrastructure*, pages 33–47. Springer.

Zeldes, A., Ritz, J., Lüdeling, A., and Chiarcos, C. (2009). ANNIS: A search tool for multi-layer annotated corpora. In *Corpus Linguistics*, pages 20–23, Liverpool, UK.