The ACoLi CoNLL Libraries: Beyond Tab-Separated Values

Christian Chiarcos, Niko Schenk
Applied Computational Linguistics (ACoLi) Lab
Goethe University Frankfurt, Germany
{chiarcos,nschenk}@em.uni-frankfurt.de

Abstract

We introduce the ACoLi CoNLL libraries, a set of Java archives to facilitate advanced manipulations of corpora annotated in TSV formats, including all members of the CoNLL format family. In particular, we provide means for (i) rule-based re-write operations, (ii) visualization and manual annotation, (iii) merging CoNLL files, and (iv) data base support.

The ACoLi CoNLL libraries provide command-line interface to these functionalities. The following aspects are technologically innovative and exceed beyond the state of the art: We support *every* OWPL (one word per line) corpus format with tab-separated columns, whereas most existing tools are specific to one particular CoNLL dialect. We employ established W3C standards for rule-based graph rewriting operations on CoNLL sentences. We provide means for the heuristic, but fully automated merging of CoNLL annotations of the same textual content, in particular for resolving conflicting tokenizations. We demonstrate the usefulness and practicability of our proposed CoNLL libraries on well-established data sets of the Universal Dependency corpus and the Penn Treebank.

Keywords: CoNLL data format, CoNLL-RDF, merging conflicting tokenization, interoperability

1. Background & Motivation

Since 1999, the Conference on Natural Language Learning (CoNLL)[1] has established a strong tradition of annually organized shared tasks within the NLP community. The addressed linguistic phenomena exhibit great diversity and included (but were not limited to) lexical semantics, semantic role labeling, dependency and discourse parsing, and coreference resolution. With their continuous progression in terms of linguistic complexity, the shared tasks reflect the maturation of statistical NLP, the dominating paradigm of computational linguistics in the 2000s. In many cases, successful participants established reference tools, and—as it allowed for comparative evaluation—the underlying formats (for training and test data) continued to be supported by succeeding NLP tools. This in fact has reinforced the global importance of the CoNLL format family and, as a result, the CoNLL format has ultimately become a de-facto standard within the language processing community.

CoNLL and other tab separated value (TSV)-based formats are widely used, and individual dialects come with considerable tool support, e.g., CoNLL-X, CoNLL-U or the formats of corpus tools like CWB (Evert and Hardie, 2011) or Sketch Engine (Kilgarriff et al., 2014). At the moment, however, there are considerable limitations with respect to

semantics: off-the-shelf database support is limited to CoNLL dialects with invariable number of columns, thus excluding semantic role labelling (since 2004) and discourse semantics (since 2015) in CoNLL,

syntax: while parsing CoNLL into an array is trivial, graph traversal and transformation on top of this requires considerable programming efforts,

tokenization: CoNLL presupposes one token (word) per line, if tokenization diverges, annotations can no longer be compared, and

interoperability: most tools are specific to one particular CoNLL dialect, positing constraints on columns and/or additional metadata as provided in comments.

We introduce the ACoLi CoNLL libraries in order to address these issues: They are applicable to every CoNLL TSV dialect, they provide flexible graph traversal and graph rewriting using existing W3C standards, they allow export into and import from off-the-shelf RDF triple stores, and support the automated merging of conflicting tokenizations. As shown in Fig. 1, the ACoLi CoNLL libraries can be used in different types of NLP architectures to complement existing NLP modules with an advanced level of interoperability: The **CoNLL Merge** package transforms and integrates *conflicting tokenizations* in CoNLL TSV, it *merges CoNLL files* by aggregating all (retokenized) columns of the source files, the **CoNLL-RDF** package yields an isomorphic rendering of CoNLL in *RDF*, thereby enabling *advanced manipulation of annotations* (graph rewriting) by SPARQL Update, *off-the-shelf database* support by RDF Triple/Quad Stores, and access with a W3C standardized *query language* (SPARQL). Both packages are implemented in Java and available under Apache 2.0 license.[2]

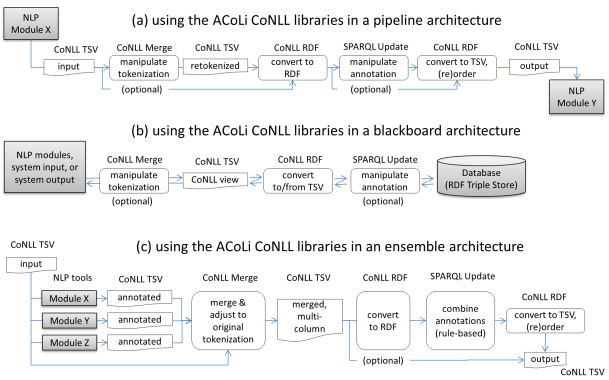

Figure 1: Using the ACoLi CoNLL libraries in different NLP architectures

[1] http:/www.signll.org/conll

[2] https://github.com/acoli-repo/conll-rdf, https://github.com/acoli-repo/conll

2. Manipulating TSV data

The **original data model** of CoNLL TSV formats is a sequence of variable-width tables, each representing a sentence, with rows in a fixed (sequential) order. Different CoNLL dialects differ in the choice and definition of columns, however, all are characterized by putting *one word per line* (OWPL), with annotations of these words stored in the respective rows. While basic manipulations such as reordering, merging, and dropping columns are simple and can be accomplished with off-the-shelf tools such as sed, advanced NLP problems often require manipulating *rows*: All CoNLL formats come with the limitation of imposing one single segmentation level on the text, but in complex NLP architectures, different modules may produce, require or produce files with deviating segmentations. Yet, tokenization strategies generally differ with respect to the research question, tokenizations can drastically *disagree*, so that multiple linguistic annotations on top of concurrent tokenizations of the same text usually require intense efforts for harmonization, as in the following phrase from the Penn Treebank (Marcus et al., 1999), resp. OntoNotes (Hovy et al., 2006):

```
a           DT        a      DT
                      19     CD
19-month    JJ        -      HYPH
                      month  NN
                      cease  NN
                      -      HYPH
cease-fire  NN        fire   NN      (wsj0655)
```

Problems multiply if an annotation inserts empty elements, e.g., the OntoNotes syntax in opposition to the OntoNotes NER. With the `CoNLLAlign` class in the **CoNLL Merge** package, we provide a fully automated solution for merging CoNLL files with concurrent tokenizations in three different modes:

default adopt the first tokenization as gold tokenization, lossless (for mismatches, add a new line after the mismatch, and mark it as an 'empty' non-word by `*RETOK*`)

-f force the first tokenization as gold tokenization (for mismatches, attach the annotations of the mismatched word to the annotations of the last line, using + as concatenator)

-split split tokens into longest common substrings, lossless (for mismatches, split annotations using the I[O]BE[S] scheme)

Figure 2 illustrates these strategies. We recommend the default option for the quantification and inspection of tokenization mismatches in preparation of NLP experiments or corpus conversion; `-f` for NLP pipelines, as this permits flexible retokenization against a target tokenization (at the price of information loss); `-split` for the fast integration of corpus data: annotations are anchored in the primary data (no artificial 'empty' elements), and the original annotation is recoverable.

CoNLL Merge builds on Myer's Diff (Myers, 1986), applied to the `WORD` (`FORM`) column, and is thus capable of tolerating spelling differences, e.g., for escaped special characters like brackets or quotes. With `-split`, it iterates on non-aligned sequences with a character-level diff, with sequences of matching characters pairs are merged into substrings.

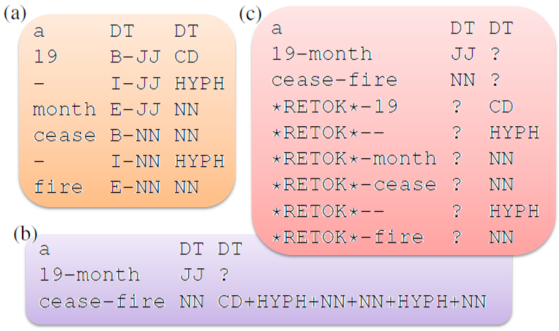

Figure 2: Merging concurrent tokenizations with **`-split`** (a), **`-f`** (b) and default options (c)

To our best knowledge, CoNLL Merge is the first publicly available tool for the fully automated, domain-independent merging of linguistic annotations of the same text with concurrent tokenizations. It does build on earlier work by Chiarcos et al. (2012), but while their implementation was specific to a complex XML standoff format with limited technological support from the wider community, we now provide an open source implementation for a popular de-facto standard for linguistic annotations in NLP.

3. Beyond Tab-Separated Values

Off-the-shelf database support for CoNLL TSV is currently limited to dialects with constant number of columns. However, this cannot be taken for granted, at least for semantic annotations, as the CoNLL representation of SRL, for example,[3] introduces an additional column *per frame instance*, i.e., sentences vary in the number of columns.

To facilitate CoNLL processing and querying, the **CoNLL-RDF** package provides a mapping from OWPL corpus formats to an isomorphic, and lossless representation in RDF (Chiarcos and Fäth, 2017). In order to provide a generic converter of CoNLL data, we must not rely on a fixed set or order of columns, but instead, expect user-provided labels. CoNLL-RDF provides a trivial of mapping comparable to CSV2RDF (Tandy et al., 2015), yet, a number of CoNLL-specific extensions are required (marked by * below):

- (*)Preserve CoNLL comments as comments, but do not interpret them.
- Assign every non-empty row a unique URI ('primary key') based on a user-provided base URI for the document, the sentence number and the word ID (or position): In the resource `https://github.com/UniversalDependencies/UD_German/blob/master/de-ud-dev.conllu`, the second word in the first sentence will receive the URI `https://github.com/`

[3] Similar problems exist with the representation of discourse relations for the 2015-2016 shared tasks.

UniversalDependencies/UD_German/blob/master/de-ud-dev.conllu#s1.2,[4] resp., :s1.2 in short.

- (*)Define every row as a word, and connect it to its successor using the NIF vocabulary (Hellmann et al., 2013)[5]: :s1.2 a nif:Word; nif:nextWord :s1.3 .
- Given a **user-provided list of column labels** (say, LEMMA for the second column in UD), we create datatype properties in the conll: namespace, and assign the word its corresponding annotation as a literal value, e.g., :s1.2 conll:LEMMA ''sein''.
- As an exception, the HEAD column is mapped to an object property ('foreign key'): :s1.2 conll:HEAD :s1.5 . This convention enables special handling of intra-sentential cross-references and is suitable for—but not restricted to—dependency syntax.
- (*)Special treatment for the user-provided label X-ARGs: If there is a column X, say, PRED, then assume that every word with a non-empty value for PRED introduces an additional column PRED-ARGi for its arguments. For every word w that has a non-empty annotation a in PRED-ARGi, and the word p_i that has the ith non-empty value in PRED, define w as argument of p_i: p_i conll:a w., e.g., :s1.2 conll:A1 :s1.1 .

In consequence, we obtain an isomorphic representation of the original CoNLL data structure in RDF which is semantically shallow,[6] but can be effectively queried, manipulated and serialized back into CoNLL using off-the-shelf RDF technology. In particular, this includes a rich infrastructure of databases, webservices, APIs, models for resource publication and linking (Chiarcos et al., 2013).

Even though it lacks formal semantics (by design), the CoNLL RDF model can also serve as a basis to transform CoNLL data into semantically well-defined formalisms such as POWLA (Chiarcos, 2012) or full-fledged NIF (Hellmann et al., 2013).

For the en-bloc conversion of CoNLL data to CoNLL-RDF, we provide the JAVA class CoNLL2RDF. Fig. 3 illustrates CoNLL-RDF sample data for the first sentence of the German UD development set, together with its rendering in CoNLL and other derived representations.

4. Advanced Graph Operations

A key advantage of an RDF representation is that a W3C-standardized query language for the flexible querying and manipulation of this data can be employed (Buil Aranda et al., 2013, SPARQL). As an example, consider querying paths in a dependency tree. SPARQL 1.1 property paths provide convenient means for complex path configurations in labeled graphs, supporting logical operations, reversal of direction and iterations of edges. While, after conversion from CoNLL TSV, dependency labels are stored as literal values of conll:EDGE, it is easy to transform them into object properties using a simple SPARQL Update statement:

```
INSERT {
  ?dep ?prop ?head .
} WHERE {
  ?dep conll:HEAD ?head .
  ?dep conll:EDGE ?edge .
  BIND(IRI(CONCAT(
    'http://universaldependencies.org/u/dep/',
?edge) AS ?prop))
}
```

As the example also illustrates, RDF resources can be defined in a way that their IRIs/URIs resolve against external resources, e.g., the dependency label nsubj yields the URL http://universaldependencies.org/u/dep/nsubj and thus a human-readable definition. For resources other than UD, one can thus directly link to *machine-readable* information and use this in a federated search, e.g., regarding linguistic annotations,[7] lexical entries from Wiktionary,[8] semantic frames,[9] multilingual word sense information,[10] or general concept knowledge.[11] After this transformation, it is now possible, for example, to retrieve all nominal subject of verbs, including those nested in conjunctions:

```
PREFIX ud:
  <http://universaldependencies.org/u/dep/>
SELECT ?verb ?nsubj
WHERE {
  ?verb conll:UPOS "VERB".
  ?verb ud:nsubj/ud:conj? ?nsubj.
  ?nsubj conll:UPOS "NOUN".
}
```

An interesting feature here is that complex graph configurations can be expressed and retrieved in a compact and human-readable way: here, an optional transition along a ud:conj edge is permitted, but not required.

SPARQL Update can not only be used for querying, but also for *manipulating* annotations. To facilitate processing data streams, the class CoNLLStreamExtractor in the **CoNLL RDF** package reads CoNLL from stdin, applies CoNLL2RDF sentence by sentence and returns valid CoNLL-RDF. In addition to this mere conversion functionality, the CoNLLStreamExtractor supports data manipulation by means of SPARQL Update: It takes as additional arguments a list of files containing SPARQL Update

[4] The UD sent_id is currently not used, because it only appears in a comment. However, future support for UD-specifics is possible.

[5] http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core

[6] Note that the conll: namespace used here is not connected with any ontology, but populated by properties as defined by the user (column labels) or in the data (values for X-ARGs columns).

[7] http://purl.org/olia

[8] http://kaiko.getalp.org/about-dbnary/

[9] http://premon.fbk.eu/

[10] http://babelnet.org/

[11] http://dbpedia.org/

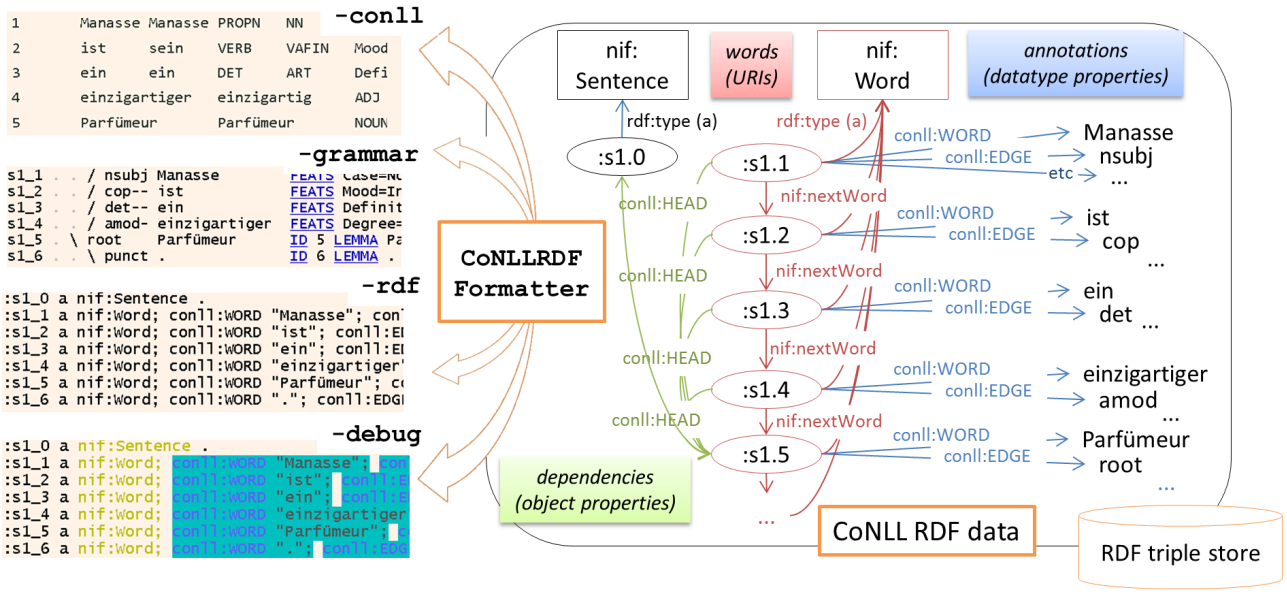

Figure 3: German sample sentence from UD, its CoNLL-RDF representation, and different serializations/visualizations

statements, introduced by the flag `-u`. Every individual file represents a module, which can be stacked to form a processing pipeline. Successively, these SPARQL Updates are applied to the individual sentences, thereby rewriting the RDF graph. We also support iterated applications, marked by an integer that can follow the respective SPARQL file in curly brackets.

In essence, the use of SPARQL 1.1 allows *unconstrained graph rewriting and enrichment*, and sample pipelines for different tasks have been developed, including, but not limited to

- combining `conll:HEAD` and dependency labels (`conll:EDGE`) into object properties,
- converting PCFG parses to dependency parses,
- transforming dependency annotations to produce UD-conformant 'semantic' dependencies,
- reducing various span annotations to the respective heads in the dependency annotation,
- stemming (using `BIND` and SPARQL string operations),
- chunking,
- rule-based dependency parsing (using a modified shift-reduce scheme), and
- supersense inference (by consulting RDF editions of VerbNet and WordNet).

5. Serializing CoNLL-RDF

CoNLL-RDF is an extensible RDF data model that can be serialized in any RDF format. The class `CoNLLRDFFormatter` in the **CoNLL RDF** package generates different serializations of CoNLL RDF data. It reads CoNLL-RDF in blocks of sentences from `stdin`, and, depending on its parameters, produces CoNLL TSV (`-conll`), a human-readable representation for dependency annotations (`-grammar`), *canonically formatted* CoNLL-RDF (default, `-rdf`), or canonically formatted CoNLL-RDF with syntax highlighting (on Un*x shells with color support, `-debug`), see Fig. 3.

Canonically formatted CoNLL-RDF has been designed as a compromise between Semantic Web standards and the low-level processability of CoNLL TSV: It is a highly constrained subset of Turtle that emulates OWPL formats.

- Sentences are separated by empty lines.
- Following eventual prefix declarations, first line defines the sentence as a `nif:Sentence`. For all except the first sentence of a corpus, the first line is preceded by a `nif:nextSentence` statement marking its position.
- The second line holds the first content word, defines it as a `nif:Word`, followed by its `conll:WORD` and other annotations, as well as a `nif:nextWord` statement pointing to the next word in the sentence. If the current word has no dependency annotation or is a `root`, its `conll:HEAD` is the sentence.
- CoNLL-RDF uses the Turtle separator ";" to enumerate the annotations assigned to a particular `nif:Word`, and "," to enumerate multiple values for the same annotation. CoNLL-RDF follows CoNLL in that all triples referring to one `nif:Word` are written in one line, concluded with ".".

The resulting format adopts many basic CoNLL conventions (comments starting with #, sentences separated by empty lines, one token per line, strictly ordered fields) and others modified (annotations separated by ; and identified by property names rather than position). Most importantly, canonically formatted CoNLL-RDF can be as easily processed with low-level string manipulations as the original CoNLL format, albeit not with an array as primary data

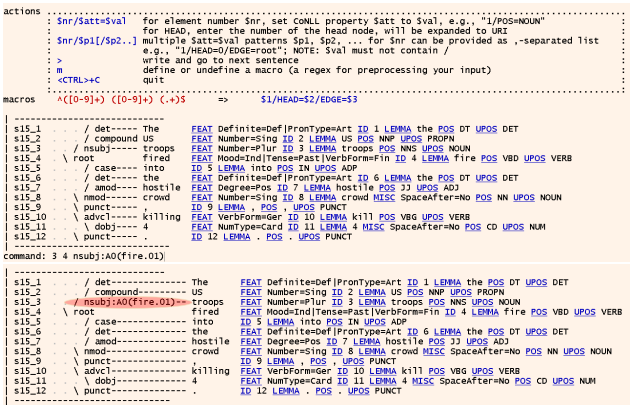

Figure 4: `CoNLLRDFAnnotator`: annotating arguments with PropBank roles (English UD corpus).

structure, but with a dictionary/hashmap. Yet, it is also possible to process canonically CoNLL-RDF with off-the-shelf RDF technology. The canonical format can be easily restored using the `CoNLLRDFFormatter`.

For NLP applications, the `CoNLLStreamExtractor` and `CoNLLRDFFormatter` can be connected in a pipeline, so that CoNLL TSV data is read from `stdin`, manipulated using SPARQL Update, and transformed back into CoNLL TSV for further processing with off-the-shelf NLP tools.

We also provide rudimentary support for manual annotation on Un*x shells with color coding: As an extension of our serialization routines, the `CoNLLRDFAnnotator` reads and writes CoNLL-RDF sentence by sentence, it visualizes it using the human-readable representation shown above, and, for every word in the sentence, it allows to overwrite every `conll:`-property with a new value. With user-provided macros (replacement rules), multiple features can be overwritten, e.g., with a pre-defined macro that allows to set `conll:HEAD` and `conll:EDGE` at the same time: The operation `3 4 nsubj:A0(fire.01)` in Fig. 4 means that the third word in the current sentence should point to the fourth as its head, with a dependency label that combines the UD dependency label with information about its PropBank frame.

6. Summary

In this paper, we introduce the ACoLi CoNLL libraries, a set of Java archives to facilitate advanced manipulations of corpora annotated in TSV formats, including all members of the CoNLL format family. These are based on our earlier research on corpus representation (Chiarcos, 2012; Chiarcos and Fäth, 2017) and structural interoperability (Chiarcos et al., 2012) with a focus on the popular CoNLL format family.

The primary goal of this effort is to facilitate interoperability between existing components in complex NLP architectures. The CoNLL Merge package provides a fully automated solution to the problem of concurrent tokenizations and leverages concurrent tokenizations of the same text by supporting fully automatized retokenization of annotated texts. The CoNLL-RDF package provides a transformation of CoNLL TSV into (and from) an isomorphic, but shallow RDF representation of CoNLL data, which facilitates database support and querying, and enables advanced manipulations of annotations CoNLL data by means of graph rewriting. We provide stream processing capabilities for these operations, by processing CoNLL and CoNLL-RDF data streams sentence by sentence, and export in different serializations, including CoNLL TSV, canonically formatted CoNLL-RDF with optional syntax highlighting, and a human-readable view for dependency annotations. In addition, a rudimentary annotation functionality is provided.

The ACoLi CoNLL libraries are designed as minimal software components that provide a layer of interoperability for glueing together heterogeneous modules and existing software components in complex NLP systems. CoNLL Merge allows to leverage tokenization differences and CoNLL-RDF permits a full rewrite of existing annotations. Beyond this, CoNLL-RDF can actually be used to implement rule-based NLP components, using SPARQL property paths and SPARQL Update for graph rewriting. As standalone application, they only provide a command-line interface to their functionalities.

With SPARQL Update, CoNLL-RDF provides a powerful, and W3C-standardized graph rewriting formalism which allows us to separate the transformation logic (SPARQL Update) from the conversion between different CoNLL dialects (CoNLLStreamExtractor, CoNLLRDFFormatter). Example transformations are provided with the release, more complete pipelines as well as rule-based NLP components developed in SPARQL Update will be subject to subsequet publications.

CoNLL Merge and CoNLL-RDF are designed to be applicable to every OWPL corpus format and thus depend on user input regarding the labels and types of columns. They impose minimal terminological constraints:

- Merging and visualization: The user has to identify the column that contains the primary data (we follow the original terminology in naming this `WORD` rather than `FORM`).
- RDF conversion: If a column `ID` is provided, it will be used to generate URIs. Columns with labels `HEAD` or X`-ARGs` are rendered as object properties ('foreign key') rather than string annotations.
- Visualization: For dependency relations and their labels, we expect the column labels `HEAD` and `EDGE`.
- RDF to TSV conversion: We export properties from the `conll:` namespace in the user-defined order.
- CoNLL RDF properties and IDs must conform to IRI conventions, i.e., the following characters are reserved: : / ? # [ ] @ $ & ' ( ) * + , ; =

No additional a priori naming conventions apply within the CoNLL-RDF core infrastructure. User- or usecase-specific SPARQL Update scripts do impose naming conventions. CoNLL Merge and CoNLL-RDF, together with sample data and sample scripts, are released under the Apache license 2.0 via our public Github repository `https://github.com/acoli-repo`.

Acknowledgments

Our research at Goethe University Frankfurt was supported by the project 'Linked Open Dictionaries (LiODi, 2015-2020)', funded by the German Ministry for Education and Research (BMBF).

7. References

Buil Aranda, C., Corby, O., Das, S., Feigenbaum, L., Gearon, P., Glimm, B., Harris, S., Hawke, S., Herman, I., Humfrey, N., Michaelis, N., Ogbuji, C., Perry, M., Passant, A., Polleres, A., Prud'hommeaux, E., Seaborne, A., and Williams, G. (2013). SPARQL 1.1 overview. Technical report, W3C Recommendation.

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: First International Conference, LDK 2017, Galway, Ireland, June 19-20, 2017, Proceedings*, pages 74–88. Springer International Publishing, Cham.

Chiarcos, C., Ritz, J., and Stede, M. (2012). By all these lovely tokens... merging conflicting tokenizations. *Language resources and evaluation*, 46(1):53–74.

Chiarcos, C., McCrae, J., Cimiano, P., and Fellbaum, C. (2013). Towards open data for linguistics: Linguistic linked data. In Alessandro Oltramari, et al., editors, *New Trends of Research in Ontologies and Lexical Resources: Ideas, Projects, Systems*, pages 7–25. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chiarcos, C. (2012). A generic formalism to represent linguistic corpora in rdf and owl/dl. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may. European Language Resources Association (ELRA).

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. University of Birmingham.

Hellmann, S., Lehmann, J., Auer, S., and Brümmer, M. (2013). Integrating NLP using linked data. In *Proc. of the 12th International Semantic Web Conference (ISWC-2013)*, pages 98–113, Sydney, Australia, Oct. Springer.

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). OntoNotes: The 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

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, Jul.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., and Taylor, A. (1999). Treebank-3. LDC Catalog No.: LDC99T42, ISBN, 1-58563-163-9.

Myers, E. W. (1986). Ano(nd) difference algorithm and its variations. *Algorithmica*, 1(1):251–266.

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