UD Annotatrix: An annotation tool for Universal Dependencies

Francis M. Tyers
School of Linguistics
НИУ ВШЭ
Moscow
`ftyers@hse.ru`

Mariya Sheyanova
School of Linguistics
НИУ ВШЭ
Moscow
`masha.shejanova@gmail.com`

Jonathan North Washington
Linguistics Department
Swarthmore College
Swarthmore, PA
`jonathan.washington@swarthmore.edu`

Abstract

In this paper we introduce the UD ANNOTATRIX annotation tool for manual annotation of Universal Dependencies. This tool has been designed with the aim that it should be tailored to the needs of the Universal Dependencies (UD) community, including that it should operate in fully-offline mode, and is freely-available under the GNU GPL licence.[1] In this paper, we provide some background to the tool, an overview of its development, and background on how it works. We compare it with some other widely-used tools which are used for Universal Dependencies annotation, describe some features unique to UD ANNOTATRIX, and finally outline some avenues for future work and provide a few concluding remarks.

1 Introduction

Once available for only a handful of languages, treebanks are becoming much more widespread. In many respects this is thanks to the activities of the Universal Dependencies (or UD, Nivre et al., 2016) community, which is an inclusive cross-linguistic consistently-annotated collection of treebanks. The collection today includes over 100 treebanks for over 54 languages, making it among the most diverse collections of freely-available openly-licensed language data.

A large proportion of the treebanks currently available through Universal Dependencies are conversions from previous annotation schemes. However, recently treebanks are being released which have been annotated from scratch, leading to the need for annotation interfaces. There are a number of existing interfaces in use for annotating UD treebanks from scratch, from the web-based such as BRAT (Stenetorp et al., 2012) and *Arborator* (Gerdes, 2013) to offline tools like *TrEd*[2] and the *TDT Editor* of the Turku Dependency Treebank (Haverinen et al., 2014).[3]

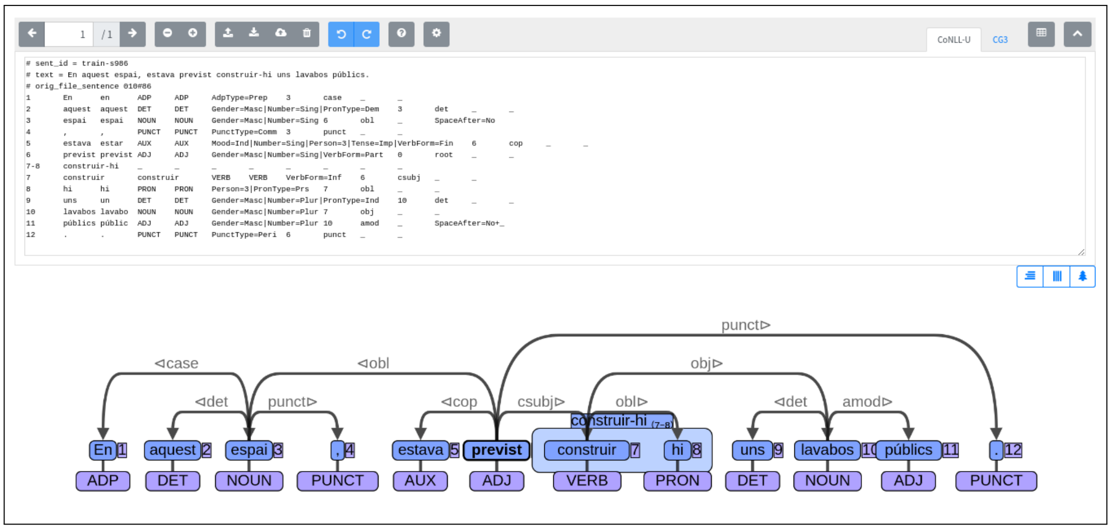
One of the things that these tools have in common is that they are not designed specifically for Universal Dependencies and so do not provide a convenient way of treating issues such as the two-level segmentation scheme (where a single surface token may be split into several syntactic words, e.g. Spanish *dímelo* 'say it to me' → dí|me|lo) and generally cannot take advantage of the annotation guidelines to provide validation feedback to the user (for example punctuation nodes may not have any dependents).

In addition, they are either based on web technologies that require some kind of server component (BRAT, *Arborator*) or are offline tools that require a number of dependencies (*TrEd* and the *TDT Editor*).

In this paper we describe UD ANNOTATRIX, a tool which can be used both online and offline, based on web technologies that is multiplatform and provides a simple interface to edit treebanks in the the CoNLL-U format[4] of Universal Dependencies.

[1] `http://www.github.com/jonorthwash/ud-annotatrix`
[2] `https://ufal.mff.cuni.cz/tred/`
[3] `https://github.com/TurkuNLP/TDT_editor`
[4] `http://universaldependencies.org/format.html`

Keywords: annotation, interface, Universal Dependencies



```
# sent_id = train-s986
# text = En aquest espai, estava previst construir-hi uns lavabos públics.
# orig_file_sentence 010#86
1       En      en      ADP     ADP     AdpType=Prep    3       case    _       _
2       aquest  aquest  DET     DET     Gender=Masc|Number=Sing|PronType=Dem    3       det     _       _
3       espai   espai   NOUN    NOUN    Gender=Masc|Number=Sing 6       obl     _       SpaceAfter=No
4       ,       ,       PUNCT   PUNCT   PunctType=Comm  3       punct   _       _
5       estava  estar   AUX     AUX     Mood=Ind|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin    6       cop     _       _
6       previst previst ADJ     ADJ     Gender=Masc|Number=Sing|VerbForm=Part   0       root    _       _
7-8     construir-hi    _       _       _       _       _       _       _       _
7       construir       construir       VERB    VERB    VerbForm=Inf    6       csubj   _       _
8       hi      hi      PRON    PRON    Person=3|PronType=Prs   7       obl     _       _
9       uns     un      DET     DET     Gender=Masc|Number=Plur|PronType=Ind    10      det     _       _
10      lavabos lavabo  NOUN    NOUN    Gender=Masc|Number=Plur 7       obj     _       _
11      públics públic  ADJ     ADJ     Gender=Masc|Number=Plur 10      amod    _       SpaceAfter=No+_
12      .       .       PUNCT   PUNCT   PunctType=Peri  6       punct   _       _
```

Figure 1: Main interface in horizontal alignment mode (see 2.4). The CoNLL-U code appears in an edit box and can be edited directly and can be hidden. The tree appears below the edit box. In addition a *table view* is supported which allows the user to view and edit the CoNLL-U data in a convenient HTML table format, with an option to toggle the visibility of individual columns.

The remainder of the paper is laid out as follows: Section 2 describes the layout and main features of the interface, Section 3 describes how it was implemented, Section 4 describes related work and how the software fits in with the general tool landscape, Section 5 describes several avenues for future work and finally Section 6 gives some concluding remarks.

2 Features

ANNOTATRIX is a tool primarily aimed at the annotation in UD of files containing up to 10,000 dependency trees. The main design principles that we have taken into account when designing ANNOTATRIX are: that the interface should display a single tree per page; the code should be stored in CoNLL-U format and able to be edited directly; the interface should as far as possible help the user by highlighting errors and proposing solutions; the tool should be zero-install and usable offline (for example for annotation sessions on flights without WiFi); and finally the features should be guided by the UD developer and user community. In taking these principles into account we have tried to prioritise the most useful functionality first with the aim of making a usable tool that can be extended based on user feedback.

2.1 Graphical editing functionality

When opening ANNOTATRIX, the user is presented with a textbox and a toolbar. The user can then either input code in a number of formats (see §2.2) into the textbox, or elect to upload a file.

Once a file is uploaded or some text is inserted, a tree appears below. The user can then click on a node, and click on another node in order to create a dependency link from parent to child. The link appears in grey, and the user can click on a direction arrow to specify a dependency relation, which can be autocompleted. There are a number of heuristics to speed up this process; for example, if the dependent is punctuation, then the `punct` relation is specified by default. It is also possible to remove dependency links by selecting them with a right click and then pressing either the DELETE or BACKSPACE key.

If different tokenisation is required, nodes can be split by right clicking and indicating where the split should be in a text box. Token indices are automatically renumbered. Nodes may also be merged, either as single tokens or as multi-word tokens. For example, given the input *verlo* ‘to see it’ in Spanish, the single token *verlo* would be split into *ver lo* and then joined into a single token span of *verlo* with two syntactic words.

Every action made in graphical mode can be reverted and made again. A detailed description of graphical editing functionality can be found on the help-page of the application.

2.2 Input formats

At the moment, Annotatrix supports five input formats (see descriptions below). They can be pasted into the text box or uploaded as a file. Code pasted into the window goes through a format detection and cleaning process. For example, if CoNLL-U format is detected but there are no tabs, a sequence of more than one space is considered a column separator. Here is a list of supported formats:

CoNLL-U: This is the standard format used in Universal Dependencies. Multilevel tokenisation and comments are supported. Null nodes and editing enhanced dependencies are currently unsupported, but support for them is planned.

CG: The input/output format used by the VISL CG3 system (Bick and Didriksen, 2015) is the native format of the Kazakh (Tyers and Washington, 2015; Makazhanov et al., 2015) and Kurdish Kurmanji (Gökırmak and Tyers, 2017) treebanks.

Stanford Dependency: A common format for specifying dependency trees in *Annodoc*[5] and the Universal Dependencies documentation. Trees can be visualised in SDParse but for editting they should be converted to CoNLL-U.

Bracket notation: Traditional bracket notation can be used for labelled dependency trees. Used in the Russian constructicon.[6] As with SDParse, visualisation is supported, but not editing.

Plain text: Plain text can be converted to CoNLL-U by a naïve spaces-and-punctuation tokenisation algorithm implemented as a regular expression.

Examples of each format can be found in Appendix A. The native format for editing is CoNLL-U, all other formats can be used for visualisation, but in order to edit the trees, they must be converted to CoNLL-U.

2.3 Text and table view

There are two ways of viewing the columns of the CoNLL-U file: CoNLL-U-formatted dependencies can be viewed in a simple HTML textarea, which allows the user to edit the file directly, as well as in a *table view* where the user is presented with a table with columns that can be shown and hidden. The table view allows better use of the available space to be made by hiding columns that might not be relevant (for example the `XPOSTAG`, `DEPS` and `MISC` columns) and also by aligning the contents of the columns. An example of table view can be seen in Figure 3.

2.4 Types of visual alignment

For very long sentences, ANNOTATRIX offers an experimental *vertical* alignment, where the tree can be viewed from top-to-bottom instead of from left-to-right. This can make better use of the available screen space by allowing more nodes to fit on the screen (the height is fixed where as the width depends on the length of the token).

In addition, we offer rudimentary support for languages with right-to-left writing systems (e.g. Hebrew, Uyghur and Arabic) to make annotating them more comfortable.[7] The full range of bidirectional (BiDi) support is not available, but we plan to add it in the future. An example of right-to-left layout with Sorani Kurdish can be found in Figure 2. ANNOTATRIX also has full Unicode support, including combining diacritics in abugida scripts.

2.5 Saving corpora

Rudimentary support for a server mode has been implemented. This mode provides support for saving user corpora on server and then accessing the saved corpora via a unique URL. This option allows the user to share their corpora with other users and makes ANNOTATRIX a simple collaboration tool.

[5] `http://spyysalo.github.io/annodoc/`

[6] `https://spraakbanken.gu.se/eng/resource/konstruktikon-rus`

[7] To our knowledge ANNOTATRIX is the only dependency-tree editting program to do this.

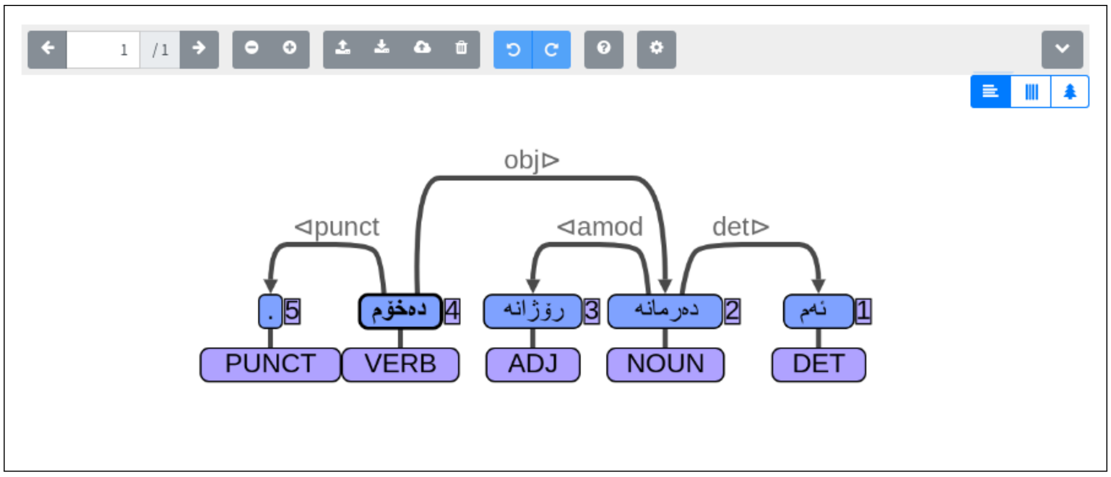


Figure 2: Support for right-to-left writing systems. Example in Sorani Kurdish reads "I take this daily medicine". This also demonstrates how the input box can be hidden to maximise space for the dependency tree.

Currently the versions of ANNOTATRIX deployed online are not linked to the server backend, but one can clone the project code and deploy it on their own web-server to use its functionality. Also, functionality is currently very limited, but more functionality is planned for the future. For example, it can be improved by adding support for tracking the editing history or by enabling the users to register and view the saved corpora on their personal page. For implementation details, see Section 3.1.

In addition to server-side saving, the entire corpus being annotated using the interface is exportable in CoNLL-U format.

2.6 Validation

ANNOTATRIX contains a number of features which help the user to annotate correctly. It offers feedback on both dependency relations and on part-of-speech tags. In the case of dependency relations, if a relation is entered which is not a universal relation (or language-specific subrelation) then the arc in the graph turns red and in table view a warning icon is displayed next to the relation with a tooltip indicating that the relation is not valid. For part-of-speech tags, the feedback is only given in table view. See Figure 3 for a demonstration of how this works.

In addition, for punctuation two rules are implemented: if punctuation is added as the head of another node or if punctuation is attached non-projectively, it is detected and the arc is turned red.

3 Implementation

3.1 Stand-alone vs. server versions

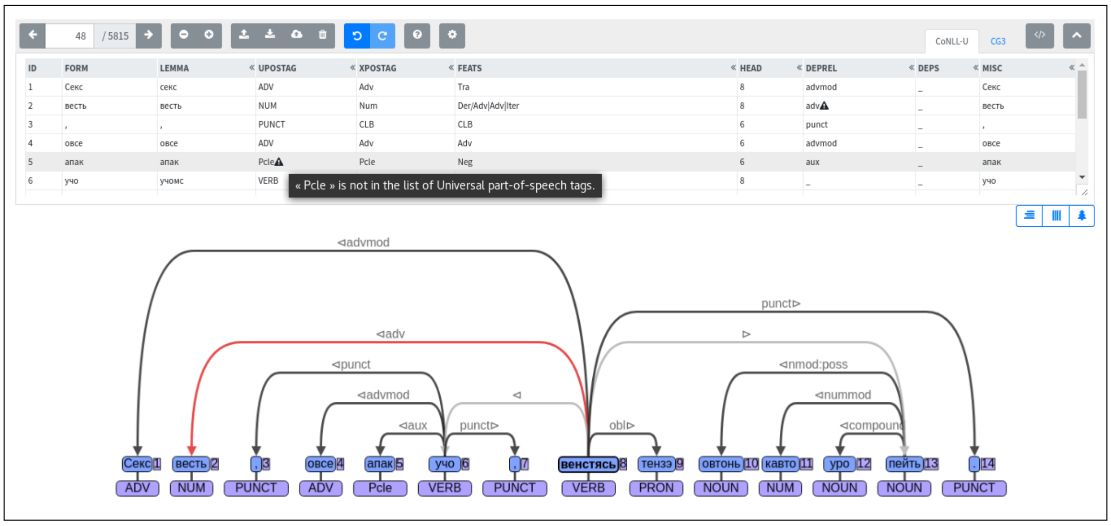
ANNOTATATRIX consists of two modules: stand-alone and server. The stand-alone part of the project supports all the functionality described in section 2, apart from saving corpora on server, whereas the server module provides additional functionality.

The stand-alone module is written in JavaScript, using jQuery and a number of dependencies described below. All the dependencies are stored locally, allowing for the offline usage of the interface. The stand-alone version stores the imported corpora in localStorage and allows for editing CoNLL-U files of up to 10,000 tokens.

The server module is written in Python 3, using the Flask web-framework. The data is passed between client and server using AJAX. As mentioned in 2.5, the server module currently has only a limited amount of functionality.

3.2 Visualisation and graphical editing

For visualising the dependency trees, we use the Cytoscape.js library (Franz et al., 2016). Cytoscape.js is an open-source JavaScript graph library primarily developed for biologists, but available to use for different purposes. It is easy-to-use and specifically designed for visualising graphs.



ID	FORM	LEMMA	UPOSTAG	XPOSTAG	FEATS	HEAD	DEPREL	DEPS	MISC
1	Секс	секс	ADV	Adv	Tra	8	advmod	_	Секс
2	весть	весть	NUM	Num	Der/Adv|Adv|Iter	8	adv⚠	_	весть
3	,	,	PUNCT	CLB	CLB	6	punct	_	,
4	овсе	овсе	ADV	Adv	Adv	6	advmod	_	овсе
5	апак	апак	Pcle⚠	Pcle	Neg	6	aux	_	апак
6	учо	учомс	VERB			8	_	_	учо

Figure 3: Screenshot showing validation features. In the table view an icon appears next to invalid values and provides a tooltip explaining the problem. In the graph view, arcs which are not labelled are first shown in grey, and then in black if they have a valid label. Arcs with an invalid label or those which are otherwise invalid (for example dependents of punctuation) are shown in red.

As dependency trees typically have much fewer nodes than biological networks and have specific layout requirements, we implemented custom functions for node and arc layout which modify the standard layouts provided by Cytoscape.js. The node layout is built based on the standard grid layout. The custom node layout allows saving horizontal space by making the cell width dependent on the token length. For the dependency links representation, the unbundled-bezier edge form was used. To avoid intersections, the height of an edge was made dependent on the distance between the nodes.

3.3 Format parsing and conversion

The main format which serves the visualisation is CoNLL-U. It is chosen as the most universal and widespread way of coding the dependency trees. For the format parsing, we used the `conllu.js` library[8] written by Magdalena Parks.

All of the other supported formats (i.e. CG, SDParse, bracket notation and plain text) are first converted to CoNLL-U, and then visualised. For unambiguous sentences in the CG format, UD Annotatrix supports visualisation and graphical editing without converting the full corpus to CoNLL-U. Each unambiguous sentence in the CG format is automatically converted to CoNLL-U for visualisation and editing support, and after the changes made in the graphical mode converted back to CG and synchronised with the graph. If the sentence is ambiguous, i.e. at least one token has several analyses, the sentence cannot be converted to CoNLL-U without loosing information. In this case, the tree is not visualised.

The format converters are tested using the the QUnit library.

3.4 Additional libraries

Large open-source libraries which Annotatrix relies on include (in the versions currently used) jQuery 3.2.1,[9] Boostrap 4.0,[10] and Font Awesome 4.7.0.[11] Additionally, preliminary support for localisation has recently been added using Mozilla's `l20n` 5.0.0,[12] and undo/redo history is implemented using a recent version of Javascript Undo Manager.[13]

[8] `https://github.com/FrancessFractal/conllu`
[9] `http://jquery.com`
[10] `http://getbootstrap.com`
[11] `http://fontawesome.io`
[12] `https://github.com/l20n/l20n.js`
[13] `https://github.com/ArthurClemens/Javascript-Undo-Manager`

4 Related work

Currently, the two tools providing the closest functionality to ANNOTATRIX are BRAT (Stenetorp et al., 2012) and *Arborator* (Gerdes, 2013). They are both web-based tools (though they require server installation). They are also both capable of processing CoNLL-U files (natively in the case of *Arborator* and with format conversion in the case of BRAT). A major difference between these two tools and ANNOTATRIX is that they both have more advanced project-management features, with users being able to curate different files and in the case of *Arborator* many useful features for classroom use of the tool (it was originally designed for classroom annotation). The current design of ANNOTATRIX has been optimised for single-file editing by a single user.

Another difference is that both BRAT and *Arborator* are annotation-scheme neutral, they offer validation support but do not offer out-of-the-box support for Universal Dependencies.

5 Future work

One of the main features that has been requested but as yet has not been implemented is the incorporation of search functionality. We envisage two modes of operation, the first could provide simple search-by-label or search-by-token/relation/etc. functionality for offline use on small treebanks. The second would be to incorporate `dep_search` (Luotolahti et al., 2017), which is an extremely powerful query language for searching in dependency parse banks.

An additional feature that we would like to integrate into ANNOTATRIX is the work of de Marneffe et al. (2017) on error finding in UD treebanks. Their current tool allows errors to be flagged, but it should be possible for trees to be fixed as well—i.e., instead of just reporting errors, a patch which fixes the error could be generated.

At the moment, the validation features of ANNOTATRIX are quite limited. It should be possible to write much more intricate rules to validate the trees; code in UD's `validate.py` and in `UDapy` (Popel et al., 2017) could be used as a basis for this, with priority going to format validation.

There is also a wide range of interface and usability improvements that are being actively worked on. These are all documented as issues in the main GitHub repository. In addition ANNOTATRIX is being actively used in annotation projects such as the Marathi (Ravishankar, 2018) and Bambara (Aplonova and Tyers, 2018) treebanks and we expect that this use will provide useful feedback in terms of bugs and feature requests.

6 Concluding remarks

This paper has presented UD ANNOTATRIX, a free/open-source tool for annotating Universal Dependencies.[14] UD ANNOTATRIX is developed for and by the community of Universal Dependencies users. The current set of features has been described, along with details on implementation, related work, and our plans going forward. It is our hope that ANNOTATRIX will streamline the workflows of many UD annotators, thereby enabling the creation of UD-annotated corpora that are larger and in a wider range of languages, and that the tool will grow and improve as more users notice bugs and request new features.

Acknowledgements

We would like to thank the Google Summer of Code and the Apertium project for supporting the development of ANNOTATRIX. In addition we would like to thank Filip Ginter and Alexandre Rademaker for extremely helpful discussions and suggestions. Tai Warner, Sushain Cherivirala and Kevin Unhammer have also contributed code. And finally, we would like to thank our users, in particular Jack Rueter and Katya Aplonova, for their helpful bug reports and encouragement.

[14] All of the source code is available online at `https://github.com/jonorthwash/ud-annotatrix`.

References

Aplonova, E. and Tyers, F. M. (2018). Towards a dependency-annotated treebank for Bambara. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, page *this volume*.

Bick, E. and Didriksen, T. (2015). Cg-3 – beyond classical constraint grammar. In *Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA*, pages 31–39. Linköping University Electronic Press, Linköpings universitet.

de Marneffe, M.-C., Grioni, M., Kanerva, J., and Ginter, F. (2017). Assessing the annotation consistency of the universal dependencies corpora. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 108–115.

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., and Bader, G. D. (2016). Cytoscape.js: a graph theory library for visualisation and analysis. 32(2):309–311.

Gerdes, K. (2013). Collaborative dependency annotation. In *Proceedings of DepLing 2013*, pages 88–97.

Gökırmak, M. and Tyers, F. M. (2017). A dependency treebank for kurmanji kurdish. In *Proceedings of the Fourth International Conference on Dependency Linguistics (DepLing, 2017)*, pages 64–73.

Haverinen, K., Nyblom, J., Viljanen, T., Laippala, V., Kohonen, S., Missilä, A., Ojala, S., Salakoski, T., and Ginter, F. (2014). Building the essential resources for Finnish: the Turku Dependency Treebank. *Language Resources and Evaluation*, 48(3):493–531.

Luotolahti, J., Kanerva, J., and Ginter, F. (2017). `dep_search`: Efficient search tool for large dependency parsebanks. In *Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa)*.

Makazhanov, A., Sultangazina, A., Makhambetov, O., and Yessenbayev, Z. (2015). Syntactic annotation of Kazakh: Following the Universal Dependencies guidelines. a report. In *3rd International Conference on Turkic Languages Processing, (TurkLang 2015)*, pages 338–350.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of Language Resources and Evaluation Conference (LREC'16)*.

Popel, M., Žabokrtský, Z., and Vojtek, M. (2017). Udapi: Universal api for Universal Dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 96–101.

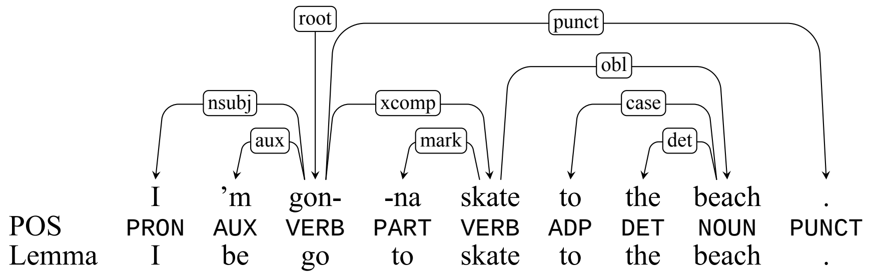
Ravishankar, V. (2018). A Universal Dependencies Treebank for Marathi. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, page *this volume*.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). BRAT: a web-based tool for NLP-assisted text annotation. In *EACL '12 Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107.

Tyers, F. M. and Washington, J. N. (2015). Towards a free/open-source Universal-dependency treebank for Kazakh. In *3rd International Conference on Turkic Languages Processing, (TurkLang 2015)*, pages 276–289.

A Demonstration of dependency annotation formats

The following sentence is rendered below in several dependency annotation formats.



root
punct
obl
nsubj
xcomp
case
aux
mark
det
I 'm gon- -na skate to the beach .
POS PRON AUX VERB PART VERB ADP DET NOUN PUNCT
Lemma I be go to skate to the beach .

A.1 CoNLL-U

```
1       I       I       PRON    _       _       3       nsubj   _       SpaceAfter=No
2       ’m      be      AUX     _       _       3       aux     _       _
3-4     gonna   _       _       _       _       _       _       _       _
3       gon     go      VERB    _       _       _       0       root    _
4       na      to      PART    _       _       5       mark    _       _
5       skate   skate   VERB    _       _       3       xcomp   _       _
6       to      to      ADP     _       _       8       case    _       _
7       the     the     DET     _       _       8       det     _       _
8       beach   beach   NOUN    _       _       5       obl     _       SpaceAfter=No
9       .       .       PUNCT   _       _       3       punct   _       _
```

A.2 CG3

```
″<I>″
        ″I″ PRON @nsubj #1->3
″<’m>″
        ″be″ AUX @aux #2->3
″<gonna>″
        ″go″ VERB @root #3->0
                ″to″ PART @mark #4->5
″<skate>″
        ″skate″ VERB @xcomp #5->3
″<to>″
        ″to″ ADP @case #6->8
″<the>″
        ″the″ DET @det #7->8
″<beach>″
        ″beach″ NOUN @obl #8->5
″<.>″
        ″.″ PUNCT @punct #9->3
```

A.3 SDParse

```
I ’m gonna skate to the beach .
nsubj(gonna, I)
aux(gonna, ’m)
xcomp(gonna, skate)
obl(skate, beach)
det(beach, the)
case(beach, to)
punct(gonna, .)
```

A.4 Bracket notation

```
[root [nsubj I] [aux ’m]  gonna [xcomp skate [obl [case to] [det the] beach]]]
```