Regenerating sentences from Universal Dependencies Structures

Guy Lapalme
RALI-DIRO, Université de Montréal
lapalme@iro.umontreal.ca

June 7, 2021

Abstract

We describe a system for regenerating sentences from English or French Universal Dependencies structures (UD). A symbolic approach is used to transform the dependency tree into a tree of constituents which is then regenerated as a sentence in the original language. We show how the output of the system can be used as a validation tool for UDs. We also describe a web-based tool for identifying UD tokens matching certain patterns.

Context of the work

We started to work on Universal Dependencies (UD) structures in the context of the Second Workshop on Multilingual Surface Realization [6] which used transformed Universal Dependencies structures as input to a surface realizer. This realizer [5] used Prolog for parsing the input structures from which a constituent tree structure was built and sent to jsRealB, a JavaScript English realizer that we have been developing for some time. The approach we followed was to first build a realizer for the original Universal Dependencies and then adapt it to take into account the transformations used for the competition: i.e., scrambling of words and abstracting relations keeping only content words.

Alessandro Sordoni, who was aware of our work asked if it was possible to use the UDs produced by a statistical parser (Stanza [9]) of an affirmative sentence to produce its negative form automatically. These modified sentences would then be used as a training corpus to *teach* the meaning of negation to a neural language mode [3]l. This system also used Prolog for parsing and creating a JSON-based tree constituent structure that was interpreted by jsRealB.

We now describe UDregenerator which uses a slightly different approach: the whole transformation process (parsing of Universal Dependencies, transformation and text generation) is performed in JavaScript and is now integrated in a web page (see Figure 1). It

allows entering Universal Dependencies structures, realization of the sentence from the information in the dependencies and comparison of the realized sentence with the original. The transformed constituent expression can also be edited for regenerating the sentence. UDREGENERATOR does not allow the modification of the UD file which should be edited externally and reloaded. The current UD is easily found in the original file using the line number or the sentence id. UDREGENERATOR can also be used as a NODE.JS module for batch processing a UD dependency file.

We initially used UDREGENERATOR to study the coverage of JSREALB of English and French by checking to what extent it was possible to recreate verbatim the original sentences. But as we will show later, this experiment also allowed us to measure to what extent the lexical information in UD is exact or complete.

We first recall the UD input format and describe the tree-based representations used by our system using the example from Figure 1. Section 3 presents the core algorithm for transforming between the these representations. Section 4 describes our experience in using UDREGENERATOR for validating UDs. We conclude with some lessons learned from this development. The appendix 6 describes a tool for identifying tokens with specific characteristics within an UD file.

1 Universal Dependencies

Universal Dependencies structures [8] (UD) is an open community effort to create cross-linguistically consistent treebank annotation for many languages within a dependency-based lexicalist framework. The latest version (2.8) [11] provides 202 treebanks in 114 languages. This data has been developed for comparative linguistics and is used in many NLP projects for developing parsers, considering UDs as gold standard.

The UD structures are provided in tab separated files in a systematic format[1]. A UD annotation is a series of lines with the following 10 fields, an empty field is indicated by an underscore (_).

`ID`	word index, starting at 1
`FORM`	how the token is written out
`LEMMA`	the lemma of the `FORM`
`UPOS`	*universal* part of speech tag of word
`XPOS`	language specific part of speech (ignored here)
`FEATS`	list of morphological features
`HEAD`	`ID` of the head or 0 for the head
`DEPREL`	name of the *universal* dependency relation to the `HEAD`
`DEPS`	*enhanced* dependency graph (not used in our work)
`MISC`	supplementary annotation, such a spacing before or after the token

[1] `https://universaldependencies.org/format.html`

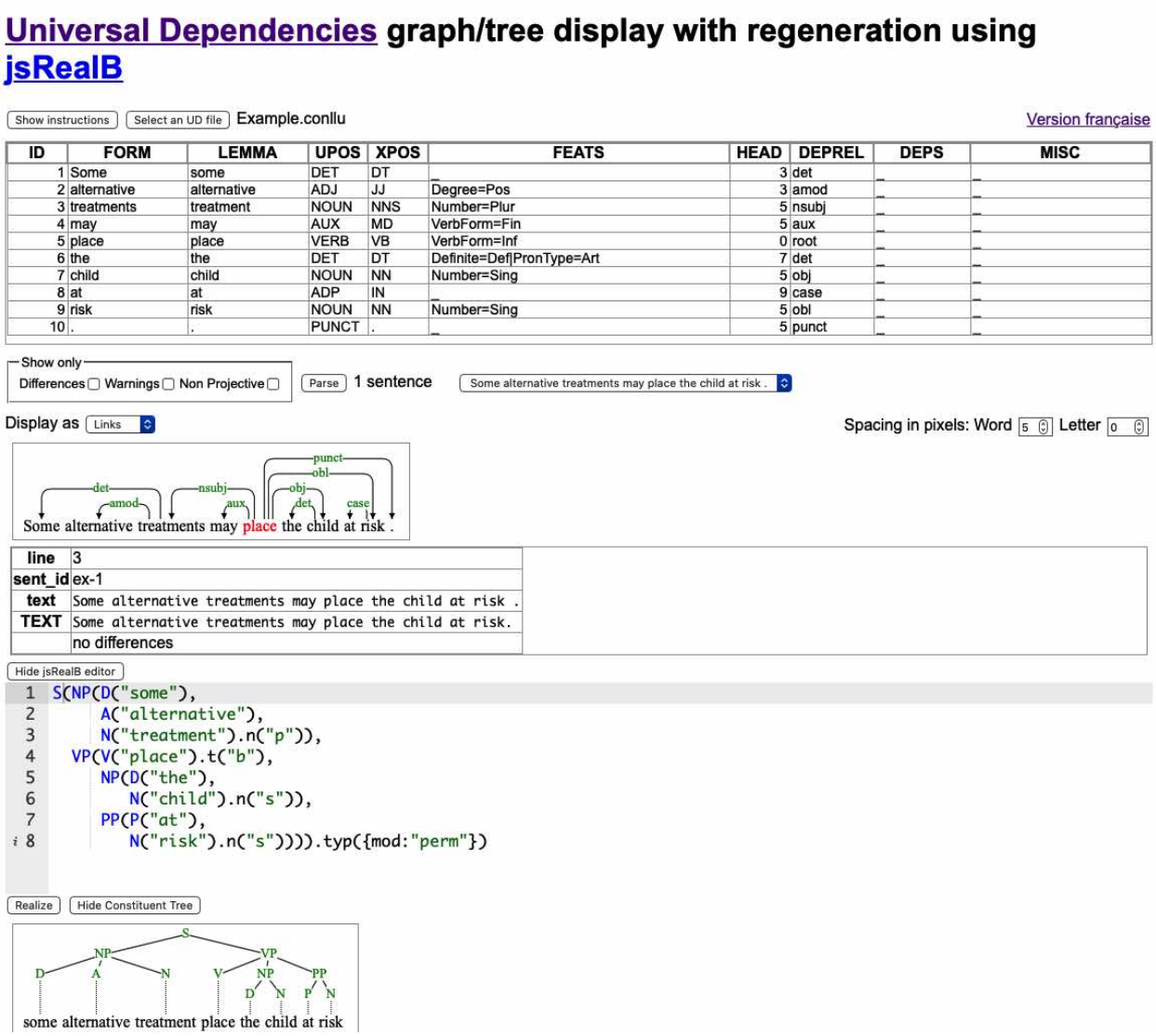


Figure 1: Web page (`http://rali.iro.umontreal.ca/JSrealB/current/demos/UDregenerator/UDregenerator-en.html`) for exploring UDs in a local file that is parsed to build a menu of their reference sentences in the middle of the page. Once a sentence is chosen, the fields of its tokens are displayed in the table at the top and the graph of its dependency links is displayed below the menu. A table below the graph shows information about this UD: the line number in the file, its `sent_id` and reference text (`text`), the regenerated text by jsRealB (`TEXT`). When there are differences between the expected text and realized tex, they are highlighted. The corresponding jsRealB expression is displayed in an editor that allows it to be changed and be re-realized. The tree of constituents corresponding to the jsRealB expression is displayed at the bottom. Checkboxes can be clicked to narrow the sentences in the menu to those for which there are differences between the reference's text and the generated sentence, those for which jsRealB issued warnings or those with non-projective dependencies.

Comments are added to the file using lines with a number sign (#). There are conventional comments such as: a line starting with `# id =` uniquely identifies a dependency structure in a file and a line starting with `# text =` indicates the text of the sentence for which the dependency structure is defined. A file can contain many UDs that are separated by a an empty line.

Many of these dependency structures are the result of manual revisions of automatic parses which are often quite difficult to check manually as there are so many details to take into account. As we will show in Section 4, regenerating from the source revealed small mistakes, most often omissions, in quite a few of the structures. It is indeed much easier to detect errors in a figure or in a generated sentence than in list of tab separated lines.

2 UDregenerator

Figure 1 shows the web-based interface of UDREGENERATOR using the UD of a simple English sentence. The table at the top shows the token fields of the selected UD in the menu with the corresponding dependency link structure in the middle.

The original UD structure is parsed to build a dependency tree which is then converted to a tree of constituents realized using JSREALB[2], a web-based English and French realizer written in JavaScript. Only the English realizer is illustrated here but there is also a web page for using the system dealing with the French version of UD. The bottom of Figure 1 shows the tree of constituents built by JSREALB after processing the UD.

An UD realizer might seem pointless, because most UD annotations are created from realized sentences either manually or automatically. As UDs contains all the tokens in their original form (except for elision in some cases), the realization can be obtained trivially by listing the `FORM` in the second column of each line.

Taking into account the tree structure, another baseline generator can be implemented using an in-order traversal of the tree and output the `form`s encountered. This method does not work for *non-projective* dependencies [4] because words, in this case, under a node are not necessarily contiguous. We use this property in our system to detect non-projective dependencies which account for about 5% of the dependencies in our corpora. But even for projective ones, different trees can be linearized in the same way. However quite often, non-projective dependencies are a symptom of badly linked nodes that should be checked. Figure 2 shows a non-projective dependency which is more easily seen when the dependencies are displayed as a tree in which some lines cross. In this case, the `HEAD` of the token *about* should be *talking* and not `what`. This is why the generated `TEXT` inserts about immediately after *what*. This figure also illustrates a type of omission error: the `FEATS` field shown in tooltip does not specify that the verb should be at the second person, so JSREALB produces the default third person, thus generating *is* instead of *are*.

What we propose in this paper is UDREGENERATOR that uses only the lemmas and the morphological and syntactic information contained in the UD features and relations to

[2] `http://rali.iro.umontreal.ca/rali/?q=en/jsrealb-bilingual-text-realiser`

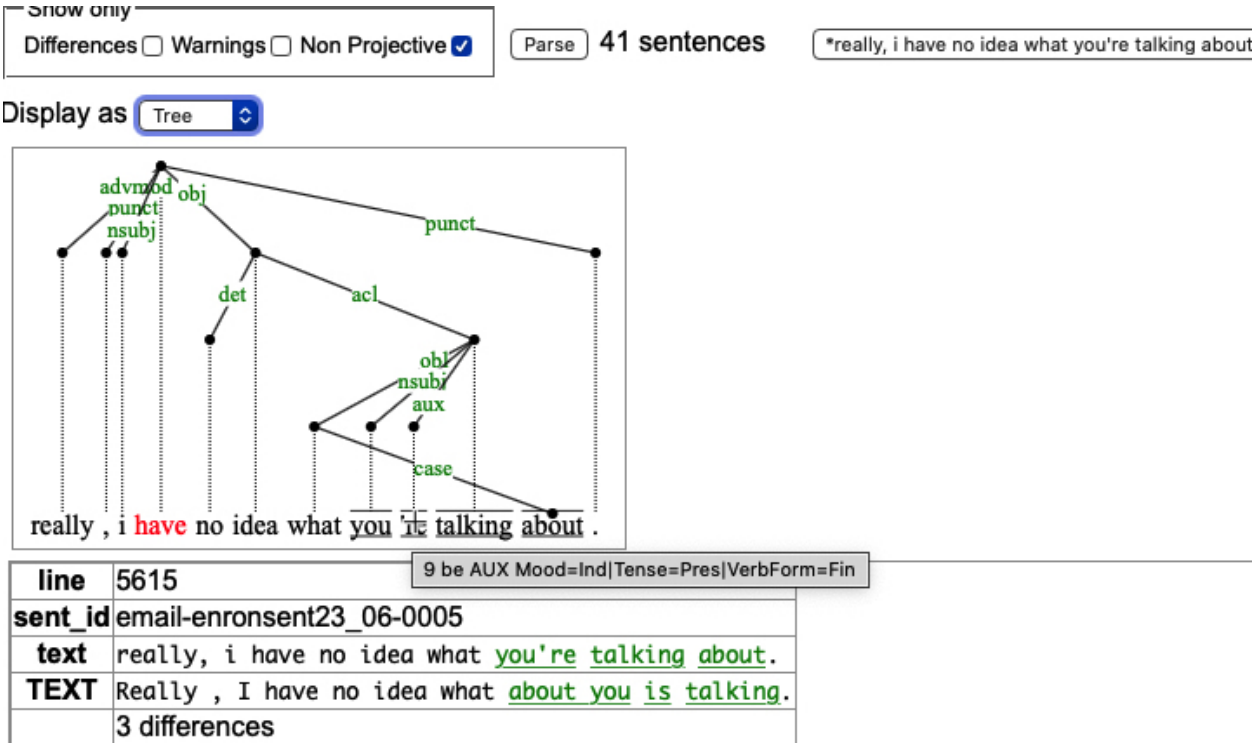


Figure 2: A non-projective UD created by a bad `HEAD` link for the word *about* which is seen in the tree display and in the bad word ordering in the generated sentence. Given that the second person is not specified in the `FEATS` field for the verb *be*, shown in the tooltip, it gives rise to an ungrammatical sentence as JSREALB uses the third person by default.

realize a sentence *from scratch* which can be compared to the original. Interesting use cases for such a realizer could be:

- UDREGENERATOR could be used as the *How to say* module of an NLG system that already provides a *What to say* module.
- An UD structure obtained by an automatic parser can be used to create variations of the original sentence using JSREALB. We have used this facility to create a training corpus of negated sentences; we have also created a training corpus of questions from affirmative sentences found on the web for improving a question-answering system [1].
- Providing help to annotators to check if the information they entered is correct by regenerating the sentence from the dependencies. This enables to catch more types of errors in the annotation; this is not foolproof, but Section 4 describes our experience with this use case.

We now briefly describe the internal representations used by our system.

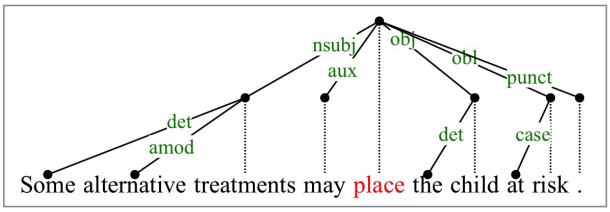
2.1 UD in JSON

The first step in UDREGENERATOR is to parse a group of lines in CONLLU format corresponding to UD structure in order to build the corresponding tree using node objects that keep track of the values of the fields. Once all nodes have been created, they are linked back to their parent using the `head` field, the root being identified by 0 in its `head` field. The features are transformed into an object to ease checking their values.

```
{deprel , upos , lemma , form , id , feats ,
 list of left children ,
 list of right children}
```

This representation keeps intact the parent-child relations and the relative ordering between the children, it also keeps track of the fact that some children occur to the left or to the right of the parent. This is easily inferred from the `ID` of each token compared with the value of its `HEAD`.

This link structure corresponds to a tree structure (see Figure 3) defined using the `head` field that refers to the `id` of the parent. This tree structure can also be displayed in the web page by selecting in the menu in the middle of the page.




```
{"deprel":"root", "upos":"VERB", "lemma":"place", "form":"place", "id":5, "head":0,
 "feats":{"VerbForm":"Inf"},
 "left":[{"deprel":"nsubj", "upos":"NOUN", "lemma":"treatment", "form":"treatments", "id":3,
          "head":5, "feats":{"Number":"Plur"},
          "left":[{"deprel":"det", "upos":"DET", "lemma":"some", "form":"Some", "id":1,
                   "head":3},
                  {"deprel":"amod", "upos":"ADJ", "lemma":"alternative",
                   "form":"alternative", "id":2, "head":3, "feats":{"Degree":"Pos"}}]},
         {"deprel":"aux", "upos":"AUX", "lemma":"may", "form":"may", "id":4, "head":5,
          "feats":{"VerbForm":"Fin"}}],
 "right":[{"deprel":"obj", "upos":"NOUN", "lemma":"child", "form":"child", "id":7,
           "head":5, "feats":{"Number":"Sing"},
           "left":[{"deprel":"det", "upos":"DET", "lemma":"the", "form":"the", "id":6,
                    "head":7, "feats":{"Definite":"Def","PronType":"Art"}}]},
          {"deprel":"obl", "upos":"NOUN", "lemma":"risk", "form":"risk", "id":9, "head":5,
                    "feats":{"Number":"Sing"},
           "left":[{"deprel":"case", "upos":"ADP", "lemma":"at", "form":"at", "id":8,
                    "head":9}]},
          {"deprel":"punct", "upos":"PUNCT", "lemma":".", "form":".", "id":10, "head":5}]}
```


Figure 3: Tree structure extracted from the dependency structure of Figure 1 and its corresponding JSON representation.

2.2 jsRealB

jsRealB[7] is a surface realizer written in JavaScript similar in principle to SimpleNLG [2] in which programming language instructions create data structures corresponding to the constituents of the sentence to be produced. Once the data structure (a tree) is built in memory, it is traversed to produce the list of tokens of the sentence.

The data structure is built by function calls whose names were chosen to be similar to the symbols typically used for constituent syntax trees[3]:

- **Terminal**: `N` (Noun), `V` (Verb), `A` (adjective), `D` (determiner) ...
- **Phrase**: `S` (Sentence), `NP` (Noun Phrase), `VP` (Verb Phrase) ...

Typically in JavaScript, identifiers starting with a capital letter are constructors not functions, however, in linguistics, symbols for constituents start with a capital letter, so we kept this convention. Features added to the structures using the dot notation, called *options*, can modify their properties. For terminals, their person, number, gender can be specified. For phrases, the sentence may be negated or set to a passive mode; a noun phrase can be pronominalized. Punctuation signs and HTML tags can also be added.

For example, in the jsRealB structure of Figure 1, plural of `treatment` is indicated with the option `n("p")` where `n` indicates the number and `"p"` the plural. Agreements within the `NP` and between `NP` and `VP` are performed automatically, although this feature is not often used in this experiment because features on each token provide, *in principle*, all the necessary morphological information.[4]

The affirmative sentence is modified to use the *permission* modal using the property `{typ({"mod":"perm"})` to be realized by the verb `may`. The modification of a sentence structure is an interesting feature of jsRealB. Once the sentence structure has been built, many variations can be obtained by simply adding a set of options to the sentences, to get negative, progressive, passive, modality and some type of questions. For example, the interrogative form What may place the child at risk? is generated by adding `"int":"was"` to the object given as parameter to `.typ()` at the end of the original jsRealB expression. This feature is not needed in this work, but it was used for creating questions from affirmative sentences to build a training corpus for a neural question-answering system [1].

3 Building the Syntactic Representation

We now describe how a UD in JSON is transformed into a Syntactic Representation (SR) which is used as input to jsRealB. The principle is to *reverse engineer* the UDs annotation guidelines[5]. This is similar to the method described by Xia and Palmer [10] to recover the

[3]See the documentation `http://rali.iro.umontreal.ca/JSrealB/current/documentation/user.html?lang=en` for the complete list of functions and parameter types.

[4]Section 4 will show that, unfortunately, this is not always the case.

[5]`https://universaldependencies.org/guidelines.html`

syntactic categories that are *projected* from the dependents and to determine the extents of those projections and their interconnections.

Although this projection process is theoretically simple, there are some peculiarities when it is applied between two predefined formalisms for which the idiosyncrasies must be taken into account. In this case, the UD relations with features being associated with each token must be mapped into jsRealB constituents with options that are applied either to a terminal or a phrase. We now give more detail on the mapping process for generating words using the morphological information associated with tokens and for generating phrases from dependency relations.

3.1 Morphology

Terminals in UD are objects whose left and right lists of children are empty. They are mapped to *terminal* symbols in jsRealB. So we transform the JSON version of the UD notation to the SR one by mapping lemma and feature names. The following table gives a few examples:

JSON fields	SR
`"upos":"NOUN", "lemma":"treatment", "feats":{"Number":"Plur"}`	`N("treatment").n("p")`
`"upos":"VERB", "lemma":"lean", "feats":{"Mood":"Ind","Tense":"Past"}`	`V("lean").t("ps")`
`"upos":"PRON", "lemma":"its", "feats":{"Gender":"Neut","Number":"Sing", "Person":"3","Poss":"Yes","PronType":"Prs"}`	`Pro("me").c("gen").pe("3") .g("n").n("s")`

As shown in the last example, we had to *normalize* pronouns to what jsRealB considers as its base form. In the morphology principles of UD[6], it is specified that *treebanks have considerable leeway in interpreting what "canonical or base form" means.* In some English UD corpora, the `lemma` of a pronoun is almost always the same as its `form`; it would have been better to use the tonic form. We decided to *lemmatize further* instead of merely copying the lemma as a string input to jsRealB so that verb agreement could eventually be performed. English UDs do not seem to have a systematic encoding of possessive determiners such as `his` which, for jsRealB at least, should be POS-tagged as a possessive determiner. These are defined as pronouns in some sentence or determiners in others, we found even cases of both encodings occurring in the same sentence. As the documentation seems to favor pronouns[7], we had to adapt our transformation process to deal with these *errors* as they occur quite often. This problem is less acute in the French UDs.

[6] `https://universaldependencies.org/u/overview/morphology.html`

[7] `https://universaldependencies.org/en/feat/Poss.html` indicates that `his` can be marked as a possessive pronoun.

What should be a `lemma` is an hotly debated subject on the UD GitHub, but there are still too many debatable lemmas such as *an*, *n't*, plural nouns etc. In one corpus, lowercasing has been applied to some proper nouns, but not all. We think it would be preferable to apply a more *aggressive* lemmatization to decrease the number of base forms for helping further NLP processing that is often dependent on the number of different types. The lexicons for jsRealB being sufficiently comprehensive for most current uses (34K lemmas for English and 53K lemmas for French), they are still unknown lemmas for specialized or informal contexts. Our experience shows that, most often, *unknown* lemmas are symptoms of errors in the lemma or the part of speech fields. Section 4.1 shows some examples encountered in the corpora.

3.2 JSON notation of UD to Syntactic Representation

The goal is to map the tree representation of the dependencies to a tree of constituents that can be used by jsRealB for realizing the sentence. According to the annotation guidelines, there are two main types of dependents: nominals and clauses which themselves can be simple or complex.

The head of a Syntactic Representation is determined by the terminal at the head of the dependencies. The system scans dependencies to determine if the sentence is negative, passive, progressive or interrogative depending on whether combinations of `aux`, `aux:pass` with proper auxiliaries (possibly modals) or interrogative `advmod` are found. When such a combination is found, then these relations are removed before processing the rest. The appropriate jsRealB sentence `typ` will be added to the resulting Universal Dependencies. For example, in Figure 1, the auxiliary `may` is removed from the tree and the sentence is marked to be realized using the *permission* modal.

All dependencies are transformed recursively; as each child is mapped to a SR, children list are mapped to a list of SR. Before combining the list of Syntactic Representations into a jsRealB constituent, the following special cases are taken into account, for English sentences:

1. a UD with a copula is most often rooted at the attribute (e.g. `mine` in Figure 4), it must be reorganized so that the auxiliary is used as the root of a verb phrase (`VP`):

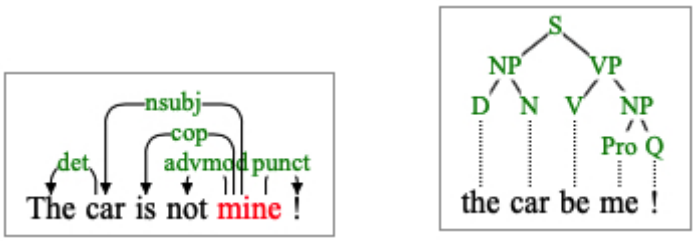


Figure 4: On the left, the dependency tree corresponding to the sentence The car is not mine!; at right, the dependency tree after transformation. jsRealB realizes the original sentence from this tree.

2. A verb at the infinitive tense is annotated in UD as the preposition `to` before the verb, so this preposition is removed before processing the rest of the tree, it is reinserted at the end;

3. An adverb (from `advmod` relation) is removed from processing the rest and added to the resulting `VP` at the end;

4. If the head is either a noun, an adjective, a proper noun, a pronoun or a number, it is processed as a nominal clause mapped to a `NP` enclosing all its children UDs.

5. If the head is a verb: check if the auxiliary `will` is present, then a future tense option will be added to the verb; in the case of the `do` auxiliary, copy feature information (tense and person) into the jsRealB options.

6. Otherwise, bundle Syntactic Representations into a sentence `S`, the subject being the first child and the `VP` being the second child.

7. Coordinate `VP`s and `NP`s must also be dealt specially because the way that jsRealB expects the arguments of a `CP` is different from the way coordinates are encoded in UDs where the elements are joined by `conj` relations. in jsRealB, all these elements must be wrapped in a global `CP`, the conjunction being indicated once at the start.

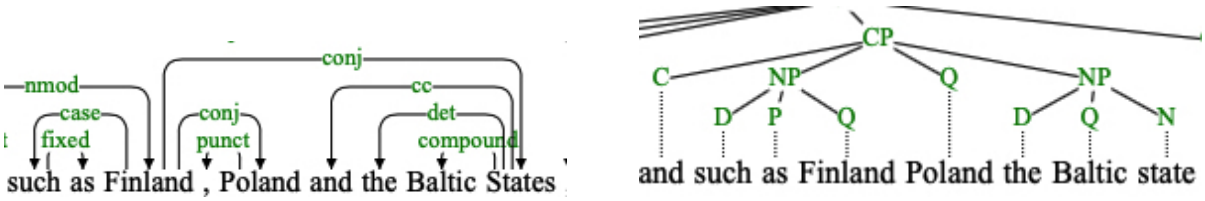


Figure 5: The graph at the left, a subgraph of the UD `w02013093` in `en_pud-ud-test.conllu`, illustrates the UD encoding of coordinated nouns Finland, Poland and the Baltic States; the right part shows the expected dependency tree by jsRealB.

We had originally implemented this tree-to-tree mechanism in Prolog (15 rules in 100 lines of commented and indented code) by reading the annotation guidelines and then refining by experimenting with the UD corpus. For UDRegenerator we *converted* this approach in JavaScript which unfortunately is much less appropriate for this type of transformation. On top of the fact that structure matching in JavaScript is more cumbersome than in Prolog, the feature that we missed the most and was more error-prone is the fact that in Prolog, it is easy to transform a tree to check for a certain condition and, when it is not met, backtracking resets it to its original state and this can occur at any nested levels. This is not the case in JavaScript where tree modifications are much more delicate to *undo*, so we had to carefully find an ordering of transformations so that tree modifications to a certain step would not have adverse effects later.

This exercise in transforming UD structures to jsRealB reveal an important difference in their level of representation. By design UD stays at the level of the form in the sentence, while jsRealB works at the constituent level. For example, in UD, negation is indicated by annotating `not` and the auxiliary elsewhere in the sentence, while in jsRealB the negation is given as an *option* for the whole sentence. So as shown above, the structure is checked for the occurrence of `not` and an auxiliary to generate the `.typ({neg:true})` option for jsRealB (see Figure 4); these dependents are then removed for the rest of the processing. Similar checks must also be performed for passive constructs, modal verbs, progressive, perfect and even future tense in order to *abstract* the UD annotations into the corresponding structure for jsRealB.

3.3 Working with French

As jsRealB can also be used for realizing sentences in French and that many UDs are available in French, we adapted for French the methodology described in the previous section. For morphology, we changed the lemmas for pronouns and numerals. Fortunately, the *ambiguity* between pronouns and determiners seldom occurs in the French UDs, so this step was more straightforward.

The transformation for clauses stays essentially the same as for English, except that there is no need to cater for the special cases for modals, future and infinitives.

4 Experiments

We experimented with version 2.8 of UD, the most recent at the time of writing. UDRegenerator can be used interactively[8], but it can also be used as a node.js module to process a whole corpus and display at a console, the results and the differences between the original text and the regenerated one.

The following subsections describe our experience running UDRegenerator on both the English and French corpora which shows that the system can handle all sentences and is quite fast: about 1 milliseconds per sentence on a commodity Mac laptop. When all lemmas of UD structure appear in the jsRealB lexicon and used with the appropriate features, UDRegenerator creates a tree and realizes the corresponding sentence. In other cases, jsRealB emits warnings so that either the unknown words can either be corrected or added to the lexicon of jsRealB for the future; in those cases, jsRealB inserts the *lemma* verbatim in the generated string which works out all right in English which is *not too morphologically rich*. But in many cases, these *erroneous lemmata* should be more closely checked. The tokens of the generated sentence are then compared with the tokens of the original text using the Levenshtein distance ignoring case and spacing. When there are differences as shown in Figure 2, they are highlighted in the output or the display; the number UDs with differences are called *#diff* in the following tables. Differences can

[8] `http://rali.iro.umontreal.ca/JSrealB/current/demos/UDregenerator/UDregenerator-en.html`

Corpus	type	#sent	#toks	#nPrj	#diff	#lerr	%regen	%terr	%nPrj
ewt	dev	2 001	25 149	44	987	219	51%	1%	2,2%
	test	2 077	25 097	41	970	174	53%	1%	2,0%
	train	12 543	204 584	462	6 780	1 698	46%	1%	3,7%
gum	dev	843	16 164	49	486	153	42%	1%	5,8%
	test	895	16 066	43	478	149	47%	1%	4,8%
	train	5 664	102 258	263	3 204	1 073	43%	1%	4,6%
lines	dev	1 032	19 170	102	664	228	36%	1%	9,9%
	test	1 035	17 765	67	645	257	38%	1%	6,5%
	train	3 176	57 372	254	2 040	702	36%	1%	8,0%
partut	dev	156	2 722	4	83	33	47%	1%	2,6%
	test	153	3 408	1	86	11	44%	0%	0,7%
	train	1 781	43 305	33	946	392	47%	1%	1,9%
pronouns	test	285	1 705	-	65	-	77%	0%	0,0%
pud	test	1 000	21 176	45	550	197	45%	1%	4,5%
Total		32 641	555 941	1 408	17 984	5 286	47%	1%	4,1%
sample		60	1 086	-	30		50%	0%	0,0%

Table 1: Statistics for the English UD corpora: for each corpus and type, it shows the numbers of sentences (#sent), of tokens (#toks) and number of non-projective dependencies (#nPrj); the number of sentences that had at least one difference with the original (#diff); the number of tokens that had at least one lexical error (#lerr); the percentages of sentences regenerated exactly (%regen), of tokens in error (%terr) and of non-projective sentences(%nPrj). The next-to-last line displays the total of these values and percentage over all sentences of the corpora. The last line shows the statistics for the sample that is studied more closely in Section 4.3.

come from limitations of jsRealB (e.g. contractions, special word ordering that cannot be generated, non-projective dependencies) or from errors or underspecification of the part-of-speech, features or head field in the UD.

As we use a symbolic approach, we do not distinguish between the training, development and test splits of a corpus, we consider them as different corpora. This allows an overall judgment on what we feel to be the precision of the informations in the UDs. The last subsection provides a more detailed analysis of a representative sample of the corpora.

4.1 English corpora

Table 1 shows statistics about the 14 English corpora that comprise 32,641 sentences of which 1,408 (4,1%) have non-projective dependencies and gave rise to 17,984 warnings. We did not use the three English ESL corpora because they do not provide any information about the lemma and the features of tokens, they only give their form and relation name.

United				
Corpus	**#occ**	**upos**	**lemma**	**feats**
EWT	93	ADJ	United	Degree=Pos
GUM	80	VERB	Unite	Tense=Past,VerbForm=Part
Lines	9	PROPN	United	Number=Sing
Partut	11	PROPN	United	
PUD	6	PROPN	United	Number=Sing
New				
Corpus	**#occ**	**upos**	**lemma**	**feats**
EWT	95	ADJ	New	Degree=Pos
GUM	80	PROPN	New	Number=Sing
Lines	21	PROPN	New	Number=Sing
Partut	3	PROPN	New	
PUD	7	PROPN	United	Number=Sing

Table 2: This table shows the different, and inconsistent across English corpora, part of speech, lemma and features associated with a two common English words used in proper names. The second column gives the number of occurrences in each English corpus.

Table 1 shows that on average about 47% of the sentences are regenerated exactly ignoring capitalization and spacing. Many of the differences are due to contractions (e.g. aint or he'll) for which jsRealB realizes the long form (is not or he will). There are two outliers: the pronouns corpus which uses a limited vocabulary and was manually designed to illustrate many variations of pronouns; in fact, we used it to design our pronoun transformations; the *lines* corpus has a high ratio of unknown lemmata some of which are *dubious*: (collapsible—expandable), &, vague as adjective, smile' and even wrote which occurs 11 times or opened, 21 times.

Looking at the results, we found that one important source of differences was the fact that in many English corpora, person and number were not given as features of verbs except for the third person singular. There are more about 11,5K instances of these in the EWT corpus, but none in the GUM corpus and about 9K in all other English corpora. As jsRealB uses the third person singular as default, the generated sentence comes out right most of the time, except when the subject is a pronoun at the first or second person or is plural.

We also discovered some inconsistencies between English corpora even for very common words. Table 2 shows occurrences of United used in United States, United Nations or United Kingdom and of New such as in New York, New England, New Delhi... In the previous version (2.7) of UDs, all United had been tagged as PROPN.

Given the fact that the jsRealB lexicon does contain the adjective United (with a capital U) or the verb Unite also with a capital, this raised warnings. A similar problem occurred for the adjective New used in New Year, New Left for which some occurrences are adjectives and others are part of a proper noun. jsRealB lexicon does not contain these lemmata with a

capital. This may seem anecdotical, but they occur quite frequently and is quite typical of inconsistency problems.

Another source of warnings is the fact that some words are tagged dubiously: there are strange conjunctions such as the following of (264 occurrences), in (181), by (162), with (142), on (99) ...

Although POS tags are consistent most of the time within a corpus, this is not the case between corpora for the same language, especially for a *non-low resource* language such as English, so some care should be used when combining these corpora in a learning scheme for a given language, unless the learning scheme does not care about number, person and POS tags.

In order to limit the number of warnings, we decided to add a few *dubious* lemmas:

- best and better were added as lemmas, although we think that the appropriate lemma should be good or well specifying the `Degree` feature: superlative (`Sup`) or comparative (`Cmp`).
- & was added as a lemma for a conjunction, but it should be and.
- in formal English, adjective and nouns corresponding to nationalities start with a capital letter (e.g. American or European), but we also had to accept the lowercase form as lemma for these[9].

4.2 French corpora

The 14 French UD corpora provide 26,586 sentences of which 1,130 (4,3 %) have non-projective dependencies. UDREGENERATOR regenerates about 53% of the sentences, which is slightly more than for the English corpora, but the overall statistics are similar between French and English.

As for English, many of the warnings were generated by *strange* part of speech tags: comme tagged (796 occurrences) as a preposition instead of adverb or conjunction, puis (225 occurrences) as conjunction instead of adverb. There a number of lemmata that were incomplete or erroneous. Here are a few examples across all French corpora:

bad part of speech : certain (343 times) is a determiner instead of an adjective; comme (796 times) is a preposition instead of a conjunction;

orthographic error : region (14 times) instead of région, inégalite (4 times) instead of inégalité, pubblicitaire (5 times) instead of publicitaire;

bad lemma : humains(6 times), performances (5 times), normes (4 times), financements, intactes or `ressources` whose lemma should be singular.

This is a good illustration of how UDREGENERATOR can help `improve` UD information.

[9]Most of these cases, have been corrected in version 2.8 of some corpora, namely EWT, following our remarks about this problem in a previous version of this paper

Corpus	type	#sent	#toks	#nPrj	#diff	#lerr	%regen	%terr	%nPrj
fqb	test	2 289	23 901	75	1 321	682	42%	3%	3,3%
gsd	dev	1 476	35 707	60	702	522	52%	1%	4,1%
	test	416	10 013	17	233	147	44%	1%	4,1%
	train	14 449	354 529	587	6 670	4 971	54%	1%	4,1%
partut	dev	107	1 870	2	64	64	40%	3%	1,9%
	test	110	2 603	1	62	68	44%	3%	0,9%
	train	803	24 122	49	506	463	37%	2%	6,1%
pud	test	1 000	24 726	17	445	460	56%	2%	1,7%
sequoia	dev	412	9 999	10	175	190	58%	2%	2,4%
	test	456	10 044	9	204	182	55%	2%	2,0%
	train	2 231	50 505	47	992	915	56%	2%	2,1%
spoken	dev	919	9 973	73	400	252	56%	3%	7,9%
	test	743	9 968	80	220	300	70%	3%	10,8%
	train	1 175	15 031	103	509	347	57%	2%	8,8%
Total		26 586	582 991	1 130	12 503	9 563	53%	2%	4,3%
Sample		60	1 233	3	37	24	38%	2%	5,0%

Table 3: Statistics for the French UD corpora: for each corpus and type, it shows the numbers of sentences (#sent), of tokens (#toks) and number of non-projective dependencies (#nPrj); the number of sentences that had at least one difference with the original (#diff); the number of tokens that had at least one lexical error (#lerr); the percentages of sentences regenerated exactly (%regen), of tokens in error (%terr) and of non-projective sentences (%nPrj). The next-to-last line displays the total of these values and percentage over all sentences of the corpora. The last line shows the statistics for the sample that is studied more closely in Section 4.3.

In both French and English corpora, we found a few instances of bad `head` links for which regeneration produces words in the wrong order (see Figure 2). The tree representation is particularly useful for checking these as there are crossing arcs. We noticed that most often this occurs in non-projective dependencies, this is why the system flags these in order so that they can be identified more easily and checked.

4.3 Sample corpora

In order to get a more precise appraisal of the quality of the UD information, we studied in detail a sample of 10 sentences from each of the 6 English and French test corpora for which we used UDRegenerator to recreate the original sentence.[10] The percentages on the last line of Tables 1 and 3 show that these samples have roughly the same characteristics

[10]These sample corpora including the equivalent jsRealB expressions are available at `http://rali.iro.umontreal.ca/JSrealB/current/demos/UDregenerator/UD-2.8/sample/`

as the whole corpus from which they were taken, except that there are no non-projective dependencies in the English sample, a small percentage anyway.

This experiment shows that jsRealB has an almost complete coverage of English and French grammatical constructs found in the corpora, except for some specialized terminology which can be easily added to the lexicon or given as quoted words that will appear verbatim in the output. We encountered only 12 unknown tokens over 1,086 in English and 4 unknown tokens over 1233 in French. In some cases (7 for English and 5 in French) jsRealB could not reproduce the exact order of some of the words in a sentence: e.g., when an adverb is inserted within a conjugated modal (e.g. would never use) or because of non-projective dependencies. In all the cases, the sentences kept their original meaning.

In English, 24 sentences were reproduced verbatim, without any modification either to the UD coding or the generated jsRealB expression. There were 5 cases of contractions (e.g., doesn't instead of does not, I've instead of I have) that jsRealB does not generate. Three cases of limitations of jsRealB because of modals being applied to noun phrases because of the transformation process limits. But we found 23 tokens (over 1,086) for which there errors or omissions in either the part of speech tags (UPOS), features or lemma. Those are very small numbers computed over only 60 sentences, the whole corpus being 490 times greater.

We also experimented with 60 sentences sampled from French corpora with the following results: 26 were regenerated verbatim without any intervention. 48 tokens (over 1233) errors or omissions in either the part of speech tags (UPOS), features or lemma. There were 5 cases of word ordering in some part of a sentence, most because of non-projective dependencies, a case of an incomplete sentence and an unusual encoding of coordination. The encoding of pronouns is especially delicate because different corpora do not use the same conventions. A case of jsRealB limitations was encountered for the verb *pouvoir*: je peux at interrogative form should be realized as puis-je and not peux-je.

This is an experiment over a very small sample (0,23%) of sentences from the French and English corpora, but we think that is shows that there is a need to recheck the information in UD as it is often used as gold standard and sometimes even used as a *mapping* source for other lower-resourced languages. We also showed that UDregenerator can be a useful tool for pointing out some eventual problems in the encoding of tokens and relations.

5 Conclusion

This work which was first motivated for exercising jsRealB in order to measure its coverage, finally made us realize that UDs while being a source of useful linguistic information, would benefit a check by trying to regenerate the sentences from the provided information. We are not aware any previous attempt to do such an experiment.

Sentence regeneration is not foolproof because different feature combinations can produce the same sentences, but we showed that in some cases it helps to pinpoint discrepancies between what is specified and the expected outcome a process similar to the Schema validation of XML files. UDregenerator is far from perfect, but it is a convenient tool for doing

some sanity checking on the lemma, part of speech, features and head fields. We hope that this work will help improve the precision of the wealth of useful information contained in UDs.

References

[1] Guillaume Le Berre, Christophe Cerisara, Philippe Langlais, and Guy Lapalme. Unsupervised multiple choice question generation for out-of-domain Q&A fine-tuning. In *submitted*, page 4p., May 2021.

[2] Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 90–93, Athens, Greece, March 2009. Association for Computational Linguistics.

[3] Arian Hosseini, Siva Reddy, Dzmitry Bahdanau, R Devon Hjelm, Alessandro Sordoni, and Aaron Courville. Understanding by understanding not: Modeling negation in language models. arXiv:2105.03519, May 2021.

[4] Sylvain Kahane, Alexis Nasr, and Owen Rambow. Pseudo-projectivity, a polynomially parsable non-projective dependency grammar. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 646–652, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics.

[5] Guy Lapalme. Realizing Universal Dependencies Structures. Internal report, `http://rali.iro.umontreal.ca/rali/sites/default/files/publis/UDSurfR.pdf`, RALI-DIRO, 10/2019 2019.

[6] Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, and Leo Wanner. The Second Multilingual Surface Realisation Shared Task (SR'19): Overview and Evaluation Results. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR), 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong, China, 2019.

[7] Paul Molins and Guy Lapalme. JSrealB: A bilingual text realizer for web programming. In *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)*, pages 109–111, Brighton, UK, September 2015. Association for Computational Linguistics.

[8] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language*

Resources and Evaluation (LREC 2016), pages 1659–1666, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[9] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages. In *ACL-2020 : System Demonstrations*, 2020.

[10] Fei Xia and Martha Palmer. Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*, 2001.

[11] Daniel Zeman, Joakim Nivre, , and *many others*. Universal dependencies 2.8.1, 2021. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

6 Appendix: Searching for combinations of tokens

UDregenerator can identify some errors or missing features in a given UD, but we found interesting to search in a file of UDs if this combination exists in other UD. It seems that many researchers use `grep` or string searching of a text editor or even special purpose scripts to check for specific combinations of features for a given token. Here are a few examples of typical searches:

- for a proper noun (`UPOS=PROPN`), the LEMMA should be the same as the `FORM`;
- when the `FEATS` contains `Typo`, the `MISC` should have a `CorrectForm`;
- if a `FORM` ends with `s`, but the `LEMMA` does not, then `FEATS` should have `Plur` as shown in Figure 6. In the identified tokens, words like `means` and `hours`, should have the feature `Number=Plur`, but given that `mechanics` and `statistics` are singular words, the lemma should be changed.

None of these combinations are foolproof, but they can easily be checked once they are identified and they can then be modified in the original file if needed. To help identify these types of feature combinations, we have set up a web page[11] to search in local UD files. Each UD field can be matched for a regular expression or its negation. It is also possible to check if a `FORM` is the same or different from the `LEMMA`. All tokens in the file that match these conditions are displayed in a table, in which it is possible to select one to get the sentence in which it occurs, the identification of the sentence (`sent_id`) and the line number in the file.

As UDgrep does not use any language specifics, it can be used to find patterns on UDs in any language. Patterns entered by the user are, of course, language specific.

[11] Available at `http://rali.iro.umontreal.ca/JSrealB/current/demos/UDregenerator/UDgrep.html`

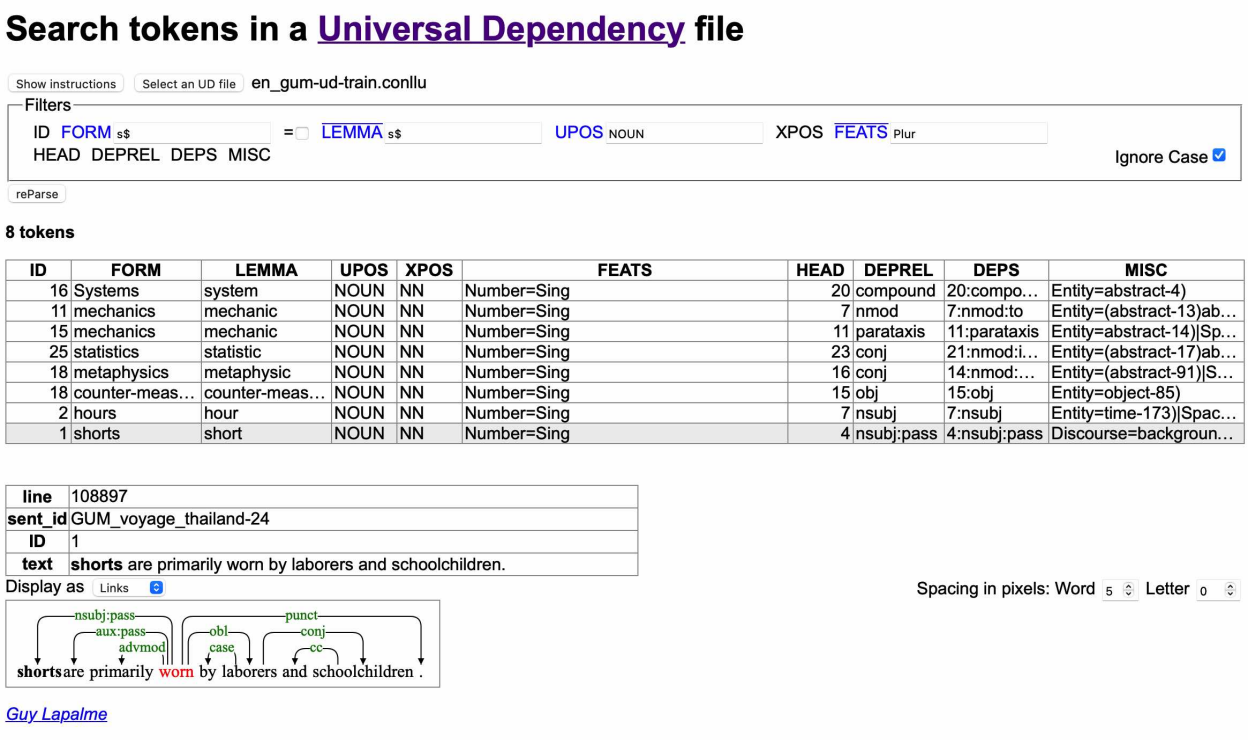


Figure 6: Identification of *curious* English nouns that end with `s`, but not their lemma and that do not contain *Plur* in their features. A colored field name shows a regular expression that should match the field, a complemented name (with an overbar) shows a regular expression that should not match. The identified tokens are shown in a table in which it is possible to select a cell to show the context of this token: sentence with the token highlighted, the id of the sentence and its line number in the file. The dependency graph of the sentence is also shown.