Validation of Universal Dependencies by *regeneration*

Guy Lapalme
RALI-DIRO / Université de Montréal
C.P. 6128, Succ. Centre-Ville
Montréal, Québec, Canada, H3C 3J7
lapalme@iro.umontreal.ca

Abstract

We describe the design and use of a web-based system for helping the validation of English or French Universal Dependencies corpora by sentence regeneration. A symbolic approach is used to transform the dependency tree into a constituency tree which is then regenerated as a sentence in the original language. The comparison between regenerated sentences and the original ones from version 2.8 of Universal Dependencies revealed some annotation errors which are discussed and give rise to suggestions for improvement.

1 Introduction

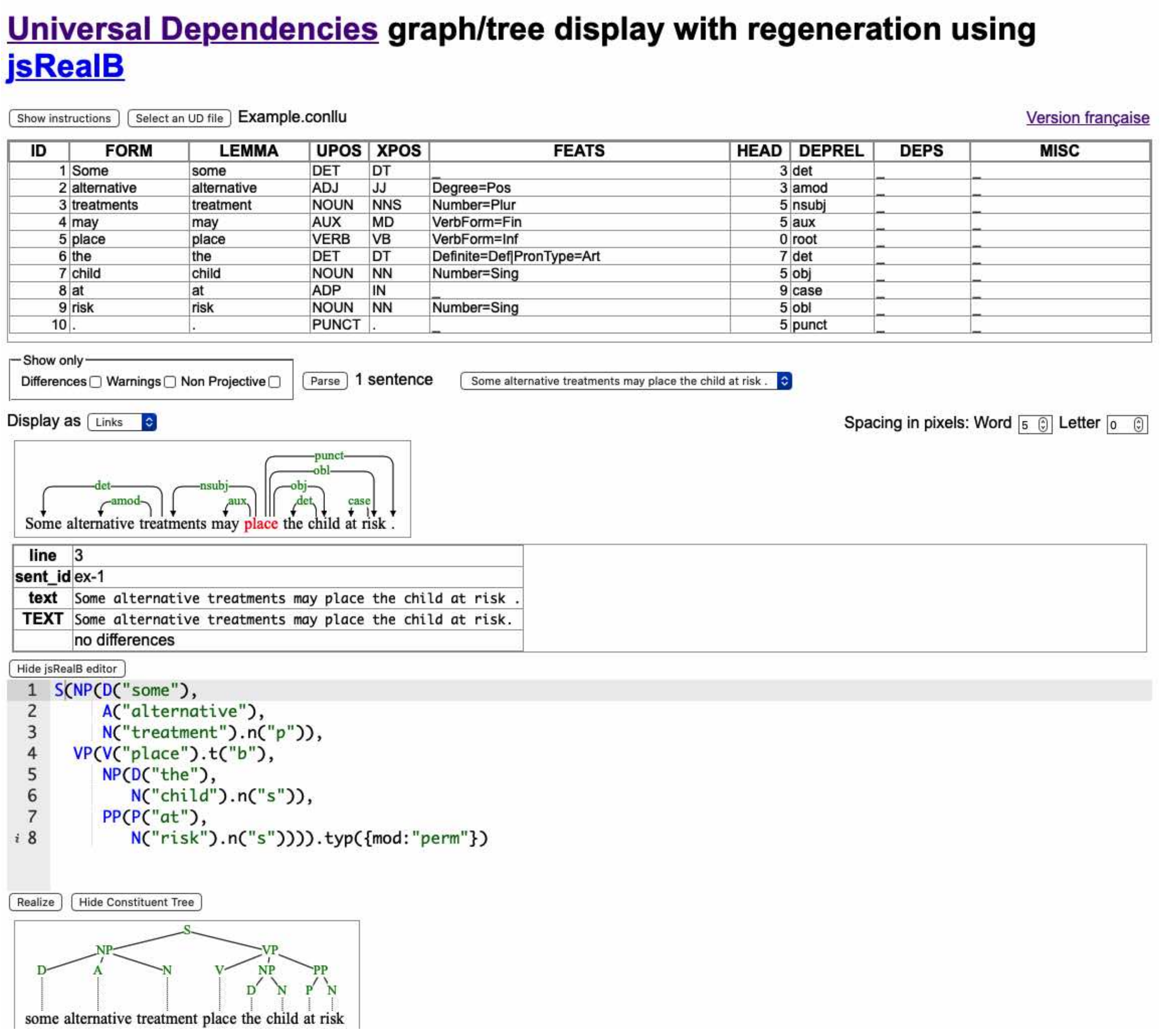
Universal Dependencies (de Marneffe et al., 2021) (UD) have been developed for comparative linguistics and are used in many NLP projects for developing parsers or machine learning systems for training and/or evaluation. The accuracy of these annotations is thus very important. Many of these dependency structure annotations are the result of manual revisions of automatic parses or mappings from other parsing formalisms (e.g. EWT was originally derived from the Penn Treebank annotations). They are often quite difficult to check manually as there are so many details to take into account. This paper describes an alternative way of looking at the UD data, using it to regenerate a sentence that can be compared with the original. As we will show in Section 3, regenerating from the source revealed small mistakes, most often omissions, in quite a few of these structures which are often considered as *gold standard*. It is indeed much easier to detect errors in a figure or in a generated sentence than in a list of tab separated lines.

This approach for helping detecting potential errors in annotations can be compared with the work of van Halteren (van Halteren, 2000) who compares the result of an automatic part-of-speech tagger with the ones in the corpus and highlights tokens in which disagreement occurs. Wisniewski (Wisniewski, 2018) detects all identical sequences of words that are annotated differently in a corpus. This is achieved by aligning, in Machine Translation parlance, the sentences with their annotation and then displaying both sequences highlighting their differences.

UDREGENERATOR (see Figure 1) is a web-based English and French realizer written in JavaScript, built using JSREALB[1]. Only the English realizer is shown here, but it is also possible to use it for checking the French corpora of UD. 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.

A UD realizer might seem pointless, because most UD annotations are created from realized sentences either manually or automatically. As UD 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 by a recursive traversal of the UD tree by first outputting the forms of the left children, then the form of the head and finally the forms of the right children.

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


Universal Dependencies graph/tree display with regeneration using jsRealB

Example.conllu

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	Some	some	DET	DT	_	3	det	_	_
2	alternative	alternative	ADJ	JJ	Degree=Pos	3	amod	_	_
3	treatments	treatment	NOUN	NNS	Number=Plur	5	nsubj	_	_
4	may	may	AUX	MD	VerbForm=Fin	5	aux	_	_
5	place	place	VERB	VB	VerbForm=Inf	0	root	_	_
6	the	the	DET	DT	Definite=Def|PronType=Art	7	det	_	_
7	child	child	NOUN	NN	Number=Sing	5	obj	_	_
8	at	at	ADP	IN	_	9	case	_	_
9	risk	risk	NOUN	NN	Number=Sing	5	obl	_	_
10	.	.	PUNCT	.	_	5	punct	_	_

1 sentence

Some alternative treatments may place the child at risk .

Display as Links

Spacing in pixels: Word 5 Letter 0

Some alternative treatments may place the child at risk .

line	3
sent_id	ex-1
text	Some alternative treatments may place the child at risk .
TEXT	Some alternative treatments may place the child at risk.
	no differences

```
S(NP(D("some"),
     A("alternative"),
     N("treatment").n("p")),
  VP(V("place").t("b"),
     NP(D("the"),
        N("child").n("s")),
     PP(P("at"),
        N("risk").n("s")))).typ({mod:"perm"})
```

some alternative treatment place the child at risk

Figure 1: Web page (`http://rali.iro.umontreal.ca/JSrealB/current/demos/UDregenerator/UDregenerator-en.html`) for exploring UD structures 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 text, they are highlighted (not shown here). The corresponding JSREALB expression is displayed in an editor that allows it to be changed and be re-realized. The constituency tree corresponding to the JSREALB expression is displayed at the bottom. Checkboxes are used to limit the sentences in the menu to those for which there are differences between the references text and the generated sentence, those for which JSREALB issued warnings or those with non-projective dependencies.

This method does not work for *non-projective* dependencies (Kahane et al., 1998) because then, words under a node are not necessarily contiguous. This property is used in our system to detect non-projective dependencies which account for about 4% of the dependencies in our corpora, which roughly corresponds to what was observed by Perrier (Perrier, 2021). But even for projective ones, different trees can be linearized in the same way. However we observed that, quite often, non-projective dependencies are a symptom of badly linked nodes that should be checked.

UDREGENERATOR realizes a sentence *from scratch* using only the lemmas and the morphological and syntactic information contained in the UD features and relations.

1.1 Constituency structure format

UDREGENERATOR transforms the UDs into a constituency structure format used as input for JSREALB (Molins and Lapalme, 2015), a surface realizer written in JavaScript similar in principle to SIMPLENLG (Gatt and Reiter, 2009) in which programming language instructions create data structures corresponding to the constituents of the sentence to be realized. Once the data structure (a tree) is built in memory, it is traversed to create the list of tokens of the sentence.

As is shown in Figure 1, the data structure is built by function calls whose names were chosen to be similar to the symbols typically used for constituent syntax trees[2]:

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

Features, called *options*, that modify some properties are added to the structures using the dot notation. For terminals, they are used to specify the person, number, gender, etc. 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*, the appropriate morphological information.[3]

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? could be 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 or for creating negations for augmenting a corpus of negative sentences for training a neural semantic analyzer.

2 Building the Syntactic Representation

The first step, not described here, is straightforward: the CONLLU input format is parsed into a JavaScript data structure for the tokens. Each token keeps information from the UD fields such as `FORM`, `LEMMA`, `FEATS`, `DEPREL` and `MISC`. The `HEAD` field is used to build a list of pointers to the tokens on its left and an another list for the tokens to the right *children*. So starting from the root, it is possible to obtain the whole UD tree. Although the position of a node, given its `ID`, is not taken into account during realization, the positions of the children relative to their parent are kept intact. We do not take into account the *absolute* node positions, because our goal is to regenerate the sentence from the *relative* positions indicated by the UD relations.

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

[3]Section 3 will show that, unfortunately, this is not always the case.

We now describe how a parsed UD is transformed into a Syntactic Representation (SR) which is used as input to jsRealB. The principle is to *reverse engineer* the UD annotation guidelines[4]. This is similar to the method described by Xia and Palmer (Xia and Palmer, 2001) to recover the 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 peculiarities to take into account when it is applied between two predefined formalisms. 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.

2.1 Morphology

UD Terminals are represented in JavaScript as tokens with no children. They are mapped to *terminal* symbols in jsRealB. So we transform the JavaScript version of the UD notation to the SR one by mapping lemma and feature names. The following table gives a few examples:

JavaScript 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", f̈eats":{"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[5], 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 UD 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 sentences or determiners in others, we even found cases of both encodings occurring in the same sentence. As the documentation seems to favor pronouns,[6] 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 UD corpora.

What should be a `lemma` is a hotly discussed subject on the UD GitHub[7], but there are still 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 to help further NLP processing that is often dependent on the number of different types. The lexica for jsRealB being sufficiently comprehensive for most current uses (34K lemmas for English and 53K lemmas for French), there 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 3.1 shows examples encountered in the corpora.

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

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

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

[7]`https://github.com/UniversalDependencies/docs/labels/lemmatization`

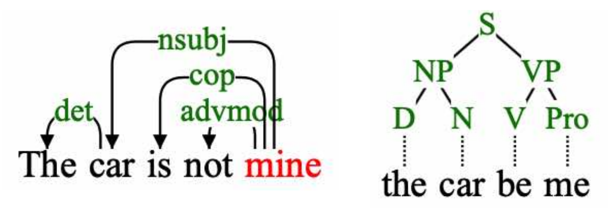
2.2 Translating the JavaScript notation of UD to Syntactic Representation

The goal is to map the JavaScript tree representation of the dependencies to a constituency tree that can be used by JSREALB to realize the sentence. According to the UD 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 2), The constituency representation must be reorganized so that the auxiliary is used as the root of a verb phrase (`VP`): This reorganization could probably be simplified with the use of the Surface Syntactic Universal



```
S(NP(D("the"),
     N("car").n("s")),
  VP(V("be").t("p").pe("3").n("s"),
     Pro("me").c("gen").pe("1")
              .g("n").n("s"))
 ).typ({neg:true})
```

Figure 2: On the left, the dependency tree corresponding to the sentence The car is not mine; the center shows the constituency tree after transformation with its JSREALB encoding on the right to realize the original sentence.

   Dependencies formalism (SUD)[8] as input as it emphasizes syntax over semantics.

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;

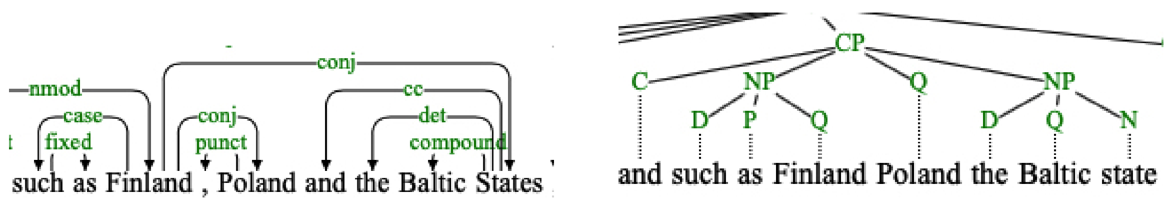
4. If the head is either a noun, an adjective, a proper noun, a pronoun or a number, it is processed as a nominal clause mapped to a `NP` enclosing all its children UD.

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, feature information (tense and person) is copied 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 in a special way because the way that JSREALB expects the arguments of a `CP` is different from the way coordinates are encoded in UD (see Figure 3) 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.

[8]\sudurl{}



nmod
case
fixed
conj
conj
punct
cc
det
compound
such as Finland , Poland and the Baltic States
CP
C
NP
Q
NP
D
P
Q
D
Q
N
and such as Finland Poland the Baltic state

Figure 3: 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 constituency tree expected by jsRealB.

This exercise in transforming UD structures to jsRealB revealed 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 2); 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. It would be interesting to check if working with SUD (Gerdes et al., 2018) would simplify the transformation process into the dependency structure.

2.3 Working with French

As jsRealB can also be used for realizing sentences in French and that many UD 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 UD corpora, 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 tense and infinitives.

3 Experiments

UDregenerator can be used interactively,[9] but it can also be used as a node.js module to process a corpus and print the differences between the original text and the regenerated one. We ran the node.js version on the French and English corpora of UD (Version 2.8), the most recent at the time of writing.[10] UDregenerator handled all sentences and is quite fast: about 1 millisecond per sentence on a commodity Mac laptop.

When all lemmas of UD structure appear in the jsRealB lexicon and are used with the appropriate features, UDregenerator creates a tree and realizes the corresponding sentence. In other cases, jsRealB emits warnings so that the unknown words can either be corrected or added to the lexicon of jsRealB; 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, they are highlighted in the output or the display; the number of UD with differences are called *#diff* in the following tables. Differences can come from limitations of jsRealB (e.g., contractions, special word ordering that cannot be generated verbatim, non-projective dependencies) or from errors or underspecification of the part-of-speech, features or head field in the UD.

For this experiment, we consider generated sentences with non-projective dependencies as *errors*. In

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

[10] Using a previous version of this tool, we detected errors in Version 2.7 that were then corrected by the maintainers of the corpora once we raised the issues to them.

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), tokens (#toks) and 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 the mean percentages over all sentences of the corpora. The last line shows the statistics for the sample that is studied more closely in Section 3.3.

the case of *legitimate* non-projectivity, the generated sentences are appropriate but with a different word order. But we also found quite a few non-projective dependencies caused by a single erroneous head link which can easily be fixed using the tree display showing crossing links. It might be interesting to explore generating the original word order by generating each token separately using only their lemma and features, but then we would lose the opportunity of checking links.

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 information in the UD. The last subsection provides a more detailed analysis of a representative sample of the corpora.

3.1 English corpora

Table 1 shows statistics on 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 the form and relation name.

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), &amp;, course as an adverb, 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

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 two common English words used in proper nouns. The second column gives the number of occurrences in each English corpus.

singular. There are about 11.5K instances of these in the EWT corpus[11], 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 UD, all United had been tagged as `PROPN`.

Given the fact that the JSREALB lexicon does not include 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 it occurs quite frequently and is typical of inconsistency problems.

Another source of warnings is the fact that some words are tagged dubiously: there are strange conjunctions such as: 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 of 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[12]:

- 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.

[11]Following our suggestion, most of these features have been added to EWT in version 2.9

[12]Many of these cases, have been corrected in version 2.8 of some corpora, namely EWT, following our remarks about this problem on version 2.7

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), tokens (#toks) and 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 the mean percentages over all sentences of the corpora. The last line shows the statistics for the sample that is studied more closely in Section 3.3.

3.2 French corpora

The 14 French UD corpora (see Table 3) 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 as a preposition instead of adverb or conjunction (796 occurrences), puis as conjunction instead of adverb (225 occurrences). There were a number of incomplete or erroneous lemmata. 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 in lemma : 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.

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

3.3 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 the 6 English and French test corpora for which we used UDREGENERATOR to recreate the original sentences.[13] The percentages on the last line of Tables 1 and 3 show that these samples have roughly the same characteristics as the whole corpus from which they were taken, except for the fact that there are no non-projective dependencies in the English sample.

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 (e.g. shippeddate, luncheonnette as noun, related or numismatic as adjective) and 4 unknown tokens over 1233 in French (e.g. boxeuse as noun or déclassifier as verb). 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 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 were errors or omissions in either the part of speech tags (UPOS), features or lemma (e.g. follow as adjective, Sir or Council as noun, with or of as conjunction). 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) had errors or omissions in either the part of speech tags (UPOS), features or lemma (e.g. mot as adjective, octobre and expire as a feminine noun, but voile (in the sense of *sail*) as a masculine noun even though the attribute is feminine). 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 it shows the need to recheck the information in UD as it is often used as a gold standard and sometimes even used as a *mapping* source for other lower-resourced languages. We also showed that UDREGENERATOR it can be a useful tool for pointing out some eventual problems in the annotation of tokens and relations.

4 Conclusion

This work made us realize that UD corpora, while being a source of useful linguistic information, would benefit from a check by trying to regenerate the sentences from the provided annotation. We are not aware of any previous attempt to perform 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. UDREGENERATOR is far from perfect, but it proved to be 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 UD corpora.

We only experimented with the French and English UD corpora because the text realizer we used only deals with these languages, but it showed the potential of detecting errors even in so-called *high resource* languages whose annotation is often considered as *golden*. It would be interesting to apply the same technique using a text realizer in other languages. Should a full-text realizer not be available for this target language, we conjecture that already checking the tokens with a conjugation or declension

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

tool might already be useful to detect some *interesting* or dubious cases. The appendix describes briefly a language independent UD exploration tool that we used to pinpoint recurrent patterns that might be symptoms of errors.

Acknowledgements

We thank François Lareau from OLST in the Linguistics Department of Université de Montréal for many fruitful discussions and suggestions about this tool and for his long-standing and continued interest in JSREALB.

References

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308, 07.

Albert Gatt and Ehud Reiter. 2009. 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. Association for Computational Linguistics.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, and Guy Perrier. 2018. SUD or Surface-Syntactic Universal Dependencies: An annotation scheme near-isomorphic to UD. In *Universal Dependencies Workshop 2018*, Brussels, Belgium, November.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. 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. Association for Computational Linguistics.

Paul Molins and Guy Lapalme. 2015. 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. Association for Computational Linguistics.

Guy Perrier. 2021. Étude des dépendances syntaxiques non projectives en français. *Revue TAL*, 62(1).

Hans van Halteren. 2000. The detection of inconsistency in manually tagged text. In *Proceedings of the COLING-2000 Workshop on Linguistically Interpreted Corpora*, pages 48–55, Centre Universitaire, Luxembourg, August. International Committee on Computational Linguistics.

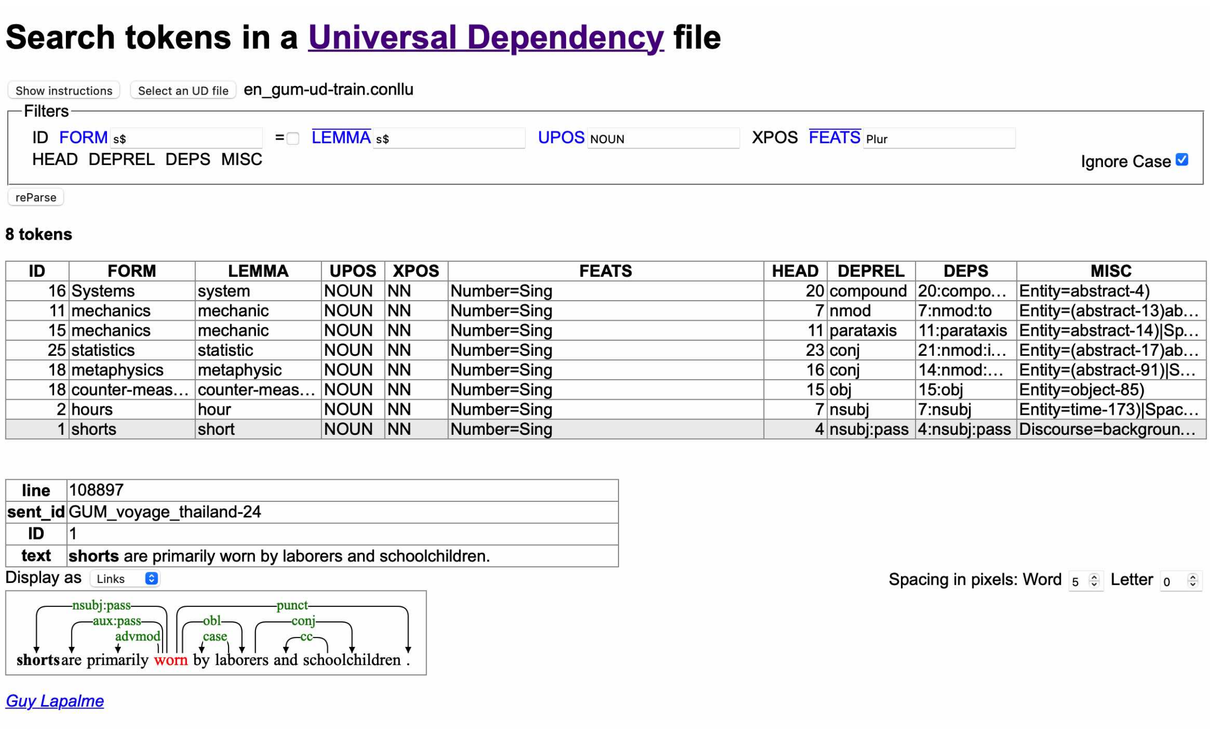
Guillaume Wisniewski. 2018. Errator: a tool to help detect annotation errors in the Universal Dependencies project. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*, pages 1–5. Association for Computational Linguistics.

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 others UD. 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. 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[14] 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. This tool is not as sophisticated as `Grew-match`[15] which defines a pattern language to allow also searching for combinations of links.

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.



Search tokens in a Universal Dependency file

Show instructions | Select an UD file | en_gum-ud-train.conllu

Filters

ID FORM s$ =☐ $\overline{\text{LEMMA}}$ s$ UPOS NOUN XPOS $\overline{\text{FEATS}}$ Plur
HEAD DEPREL DEPS MISC

Ignore Case ✓

reParse

8 tokens

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
16	Systems	system	NOUN	NN	Number=Sing	20	compound	20:compo…	Entity=abstract-4)
11	mechanics	mechanic	NOUN	NN	Number=Sing	7	nmod	7:nmod:to	Entity=(abstract-13)ab…
15	mechanics	mechanic	NOUN	NN	Number=Sing	11	parataxis	11:parataxis	Entity=abstract-14)|Sp…
25	statistics	statistic	NOUN	NN	Number=Sing	23	conj	21:nmod:i…	Entity=(abstract-17)ab…
18	metaphysics	metaphysic	NOUN	NN	Number=Sing	16	conj	14:nmod:…	Entity=(abstract-91)|S…
18	counter-meas…	counter-meas…	NOUN	NN	Number=Sing	15	obj	15:obj	Entity=object-85)
2	hours	hour	NOUN	NN	Number=Sing	7	nsubj	7:nsubj	Entity=time-173)|Spac…
1	shorts	short	NOUN	NN	Number=Sing	4	nsubj:pass	4:nsubj:pass	Discourse=backgroun…

line	108897
sent_id	GUM_voyage_thailand-24
ID	1
text	**shorts** are primarily worn by laborers and schoolchildren.

Display as Links

Spacing in pixels: Word 5 Letter 0

Guy Lapalme

Figure 4: 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.

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

[15]`http://match.grew.fr`