Realizing Universal Dependencies Structures

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

October 10, 2019

Abstract

We first describe a surface realizer for Universal Dependencies (UD) structures. The system uses a symbolic approach to transform the dependency tree into a tree of constituents that is transformed into an English sentence by an existing realizer. This approach was then adapted for the two shared tasks of SR'19. The system is quite fast and showed competitive results for English sentences using automatic and manual evaluation measures.

This document is an extended version of a paper presented at the Second Workshop on Multilingual Surface Realization [3]. It presents Universal Dependencies and the task in more details and it gives the full numerical scores obtained by our system.

1 Introduction

The goal of this work is to develop a surface realizer for Universal Dependencies structures [6] (UD), an open community effort to create cross-linguistically consistent treebank annotation for many languages within a dependency-based lexicalist framework. The latest version (2.4) provides 146 treebanks in 83 languages. This data has been developed for comparative linguistics and is used in many NLP projects for developing parsers. In fact, most of this data is the result of manual revisions of automatic parses.

The UD structures are provided in tab separated files in a well-defined format[1]. A UD annotation is a series of lines with the following fields.

`ID`	word index
`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 (abbreviated here)
`HEAD`	`ID` of the head or 0 for the head
`DEPREL`	name of the *universal* dependency relation to the `HEAD`

Figure 1 shows an annotated English sentence in UD with the corresponding linked and tree representations[2]. The tree structure is defined using the `HEAD` field that refers to the `ID` of the parent.

```
# sent_id = weblog-juancole.com_juancole_20051126063000_ENG_20051126_063000-0020
# text = His mother was also killed in the attack.
1   His      he       PRON   PRP$   Gender=Masc|...   2   nmod:poss
2   mother   mother   NOUN   NN     Number=Sing       5   nsubj:pass
3   was      be       AUX    VBD    Mood=Ind|...      5   aux:pass
4   also     also     ADV    RB     _                 5   advmod
5   killed   kill     VERB   VBN    Tense=Past|...    0   root
6   in       in       ADP    IN     _                 8   case
7   the      the      DET    DT     Definite=Def|...  8   det
8   attack   attack   NOUN   NN     Number=Sing       5   obl
9   .        .        PUNCT  .      _                 5   punct
```

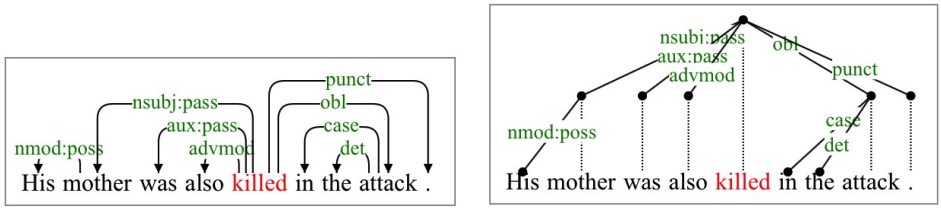


Figure 1: A Universal Dependencies in CONLLU format corresponding to the sentence given in the line starting with `# text =`.

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

[2] The figures were created using this page:`http://www-labs.iro.umontreal.ca/~lapalme/ShowUD/`

This work was prompted by the Surface Realization Shared Task 2019 (SR'19)[3] in conjunction with Second Workshop on Multilingual Surface Realization (MSR'19)[4]. We first present UD-SurfR (Universal Dependency Surface Realizer), the UD realizer before explaining its adaptation to the two tasks of SR'19 in Section 4.

Written in Prolog, UD-SurfR parses the original UD structure and builds the corresponding dependency tree which is then converted to a tree of constituents realized using jsRealB[5], a web-based English and French realizer written in Javascript; only the English realizer is used here because we worked only on the English corpora of UD. One of the benefits of this work for us has been the improvement of jsRealB by making it more robust to different kinds of input. jsRealB had been designed to ease the manual input to the realizer, but taking as input the result of another program can be quite different. Given the fact that the input and output representations are trees, Prolog seemed a natural symbolic of choice for a tree to tree transformation engine.

We did not find any text realizer that takes UD annotations as input except for Ranta and Kolachina [7] who present an algorithm to transform many UDs into Grammatical Framework structures from with English sentences can be generated.

A UD realizer might seem pointless, because UD annotations are created from realized sentences. 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 (see Section 9 for a ten line Python implementation of this idea). Unfortunately, this method does not work for *non-projective* dependencies [2] because words, in this case, under a node are not necessarily contiguous, see Section 8 for two examples. We use this property in a UD dependency display [6] 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; Section 7 illustrates this case: the linearizations of two different trees produce the same string.

What we propose in this paper is a *full* realizer that uses only the lemmas and the syntactic information contained in the UD to create the final sentence from scratch which can be compared to the original. The linear ordering of the tokens is extracted from the tree structure given by the `HEAD` links (column 7) of the UD. We can imagine two interesting uses for such a realizer:

- Should a *What to say* module of an NLG system produce UD structures, then UD-SurfR could be used as the *How to say* module.

- 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

[3] `http://taln.upf.edu/pages/msr2019-ws/SRST.html`
[4] `http://taln.upf.edu/pages/msr2019-ws/`
[5] `http://rali.iro.umontreal.ca/rali/?q=en/jsrealb-bilingual-text-realiser`
[6] `http://www-labs.iro.umontreal.ca/~lapalme/ShowUD/`

of errors in the annotation; this is not foolproof, but it is easier to detect a *strange* sentence than a bad link buried in lines of dependencies. Section 7 shows an example that we encountered in our development, where the automatic realization revealed an error in the original annotation. The error was later confirmed by the maintainer of the corpus; it had been corrected in the subsequent version of the corpus, but not in the version used for SR'19.

Over the last two years, we have developed Γω-Φ[7] using a similar approach for verbalizing Abstract Meaning Representation structures in English. UD trees proved to be somewhat simpler to process than AMR graphs as they contain all the words and punctuation, AMRs networks being more abstract.

The next section shows the tree representations used by our system using the example from Figure 1. We then show the output produced by the realizer on a more complex example used in the SR'19 task description. Section 3 presents the core algorithm for transforming between the first two representations. Section 4 describes how UD-SurfR was modified for the SR'19 tasks. Section 5 gives the results of the evaluation obtained using the evaluation scripts provided with the task; it also present the result of the manual evaluation performed on a subset of the sentences. We conclude with some lessons learned from this development.

2 Representations

Table 1 shows representations to go from the UD shown in Figure 1 to an English sentence. Row 1 shows the Prolog structure corresponding to the tree which is transformed into the Deep Syntactic Representation (DSR) shown in Row 2. Row 3 shows the Surface Syntactic Representation (SSR) which is used by jsRealB to realize the sentence shown in Row 4.

2.1 UD in Prolog

The first step is to parse a group of lines in CONLLU format corresponding to UD structure and to build the corresponding tree. The root is easily identified (0 in field 7). Its children are found by looking for lines that have the root as `HEAD`. Each child is then taken as root of the subtree and recursively parsed and transformed in the following format:

`[`*`DEPREL>LR`*`, [`*`UPOS`*`:`*`LEMMA`*` | `*`FEATS`*`] | `*`children`*`]`

`LR` is either `l` or `r` depending if the relation is to the left or the right of the `HEAD`. In Prolog, "|" separates the start of a list within brackets from the rest of the list which can be empty.

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

[7] `https://github.com/rali-udem/gophi`

1	Universal Dependencies in Prolog	`[root>r,[verb:"kill",tense:past,verbform:part,voice:pass],`
		`[aux:pass>l,[aux:"be",mood:ind,number:sing,person:3,tense:`
		`past,verbform:fin]],`
		`[obl>l,[noun:"attack",number:sing],`
		`[det>r,[det:"the",definite:def,prontype:art]],`
		`[case>r,[adp:"in"]]],`
		`[nsubj:pass>l,[noun:"mother",number:sing],`
		`[nmod:poss>r,[pron:"he",gender:masc,number:sing`
		`,person:3,`
		`poss:yes,prontype:prs]]],`
		`[punct>l,[punct:".",lin:1]],`
		`[advmod>l,[adv:"also"]]]`

2	Deep Syntactic Representation	`s(vp(ls(v("kill")*t("ps"),`
		`adv("also")),`
		`np(d("my")*pe(3)*ow("s")*n("s")*g("m")*g("m"),`
		`n("mother")*n("s")),`
		`pp(p("in"),`
		`np(d("the"),`
		`n("attack")*n("s")))))*typ({pas:true})*a(".")`

3	Surface Syntactic Representation	`S(VP(V("kill").t("ps"),`
		`Adv("also"),`
		`NP(D("my").pe(3).ow("s").n("s").g("m").g("m"),`
		`N("mother").n("s")),`
		`PP(P("in"),`
		`NP(D("the"),`
		`N("attack").n("s"))))).typ({pas:true}).a(".")`

4	English	`His mother was killed also in the attack.`

Table 1: Representations used in the transformation of the Universal Dependencies in CONLLU format in Figure 1 to the sentence shown in row 4.

the value of its `HEAD`. This is useful in some cases for realizing compounds and complements before or after the head.

2.2 Deep Syntactic Representation

The DSR is an intermediary Prolog structure that corresponds to the constituency tree of the realized sentence. A Definite Clause Grammar (DCG) transforms this structure into the SSR (described in the next subsection). In principle, it would have been possible to create the SSR directly, but it proved more convenient to use an intermediary because identifiers in Prolog starting with a capital letter indicate a variable, the dot (`.`) is dealt specially in the SWI-Prolog version we use and we sometimes build some terms incrementally using `ls` that are then merged by the DCG process. The creation of the DSR from the UD in Prolog, which is the core part of the system, is described in Section 3.

2.3 Surface Syntactic Representation

jsRealB[5] is a surface realizer written in Javascript similar in principle to SimpleNLG [1] 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.

This data structure is built by function calls whose names are the same as the symbols usually used for classical syntax trees: for example, `N` to create a *noun* structure, `NP` for a *noun phrase*, `V` for a *verb*, `D` for a *determiner*, `S` for a *sentence* and so on. Options added to the structures using the dot notation can modify the values according to what is intended.

The jsRealB syntactic representation is patterned after classical constituent grammar notations. For example,

```
S(NP(D("a"),N("woman")).n("p"),
  VP(V("eat"),
     NP(D("the"),A("red"),NP("apple"))).t("ps"))
```

is the jsRealB specification for `Women ate the red apple.` Plural is indicated with the option `n("p")` where `n` indicates number and `"p"` plural. The verb is conjugated to past tense indicated by the option tense `t` with value `"ps"`. Agreement within the `NP` and between `NP` and `VP` is performed automatically.

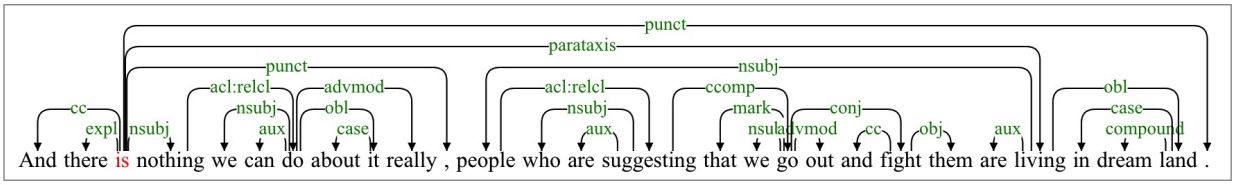
jsRealB is aimed at web developers that want to produce web pages from data[8]. It takes care of morphology, declension and conjugation to create well-formed texts. Some options allow to add HTML tags to the realized text. An interesting feature of jsRealB is the fact that once the sentence structure has been built, many variations can be obtained just by adding a set of options to the sentences, to get negative, progressive, passive, modality and some type of questions; SimpleNLG provides a similar facility. For example, adding `.typ({neg:true,pas:true,mod:"poss"})` to the previous jsRealB structure will be realized as `The red apple cannot be eaten by women.`

Row 3 of Table 1 is the jsRealB structure that is realized as the bottom part of the table. The structure of constituents written as an active sentence was finally realized as a passive one, the original complement becoming the subject. The verb was also conjugated to the past tense. This was made possible by the options given to jsRealB.

Table 2 shows the linked representation of the UD, its Prolog representation and the DSR for the example used in the description of the SR'19 task[9] and the sentence realized by jsRealB. In the generated sentence, `fights` should have been `fight`, but this happens because, in the subordinate sentence, the subject of `fight` is not given (and is not inferred by jsRealB), so it is set to the default third person.

[8]Tutorial and demos are available at `http://rali.iro.umontreal.ca/JSrealB/current/documentation/in_action/README.html`

[9]`http://taln.upf.edu/pages/msr2019-ws/SRST.html`




```
[root>r,[verb:"be",mood:ind,number:sing,person:3,tense:pres,verbform:fin],
        [cc>l,[cconj:"and"]],
        [expl>l,[pron:"there"]],
        [nsubj>r,[pron:"nothing",number:sing],
                 [acl:relcl>r,[verb:"do",verbform:inf],
                              [nsubj>l,[pron:"we",case:nom,number:plur,person:1,prontype:prs]],
                              [aux>l,[aux:"can",verbform:fin]],
                              [obl>r,[pron:"it",case:acc,gender:neut,number:sing,person:3,prontype:prs],
                                     [case>l,[adp:"about"]]],
                              [advmod>r,[adv:"really"]]]],
        [punct>r,[punct:","]],
        [parataxis>r,[verb:"live",tense:pres,verbform:part],
                     [nsubj>l,[noun:"people",number:plur],
                              [acl:relcl>r,[verb:"suggest",tense:pres,verbform:part],
                                           [nsubj>l,[pron:"who",prontype:rel]],
                                           [aux>l,[aux:"be",mood:ind,tense:pres,verbform:fin]],
                                           [ccomp>r,[verb:"go",mood:ind,tense:pres,verbform:fin],
                                                    [mark>l,[sconj:"that"]],
                                                    [nsubj>l,[pron:"we",case:nom,number:plur,person:1,prontype:prs]],
                                                    [advmod>r,[adv:"out"]],
                                                    [conj>r,[verb:"fight",mood:ind,tense:pres,verbform:fin],
                                                            [cc>l,[cconj:"and"]],
                                                            [obj>r,[pron:"they",case:acc,number:plur,
                                                                    person:3,prontype:prs]]]]]],
                     [aux>l,[aux:"be",mood:ind,tense:pres,verbform:fin]],
                     [obl>r,[noun:"land",number:sing],
                            [case>l,[adp:"in"]],
                            [compound>l,[noun:"dream",number:sing]]]],
        [punct>r,[punct:"."]]]
```

```
sp(c("and"),
   sp(pro("there"),
      vp(v("be")*t("p")*pe(3),
         np(pro("nothing")*n("s"),
            sp(pro("we")*n("p")*pe(1),
               vp(ls(v("do"),
                     adv("really")),
                  pp(p("about"),
                     pro("me")*n("s")*pe(3)*g("n"))))*typ({(mod):"poss"}))*a(","),
         sp(np(n("people")*n("p"),
               sp(pro("who"),
                  vp(v("suggest")*t("p"),
                     sp(c("that"),
                        sp(pro("we")*n("p")*pe(1),
                           vp(ls(v("go")*t("p"),
                                 adv("out")),
                              sp(c("and"),
                                 ls(v("fight")*t("p"),
                                    pro("me")*n("p")*pe(3))))))))*typ({prog:true})),
            vp(v("live")*t("p"),
               pp(p("in"),
                  np(n("dream")*n("s"),
                     n("land")*n("s")))))*typ({prog:true})))*a(".")
```

`And there is nothing we do really about it, people who are suggesting that we go out and fights them are living in dream land.`

Table 2: The dependency tree shown at the top is parsed into the nested list shown in the second line; it is then transformed into Deep Syntactic Representation shown in the third line. Using a DCG, it is transformed into a Surface Syntactic Representation (not shown here) given to jsRealB to realize the sentence shown at the bottom which can be compared with the original sentence given at the top.

3 Building the Deep Syntactic Representation

We now describe how a UD in Prolog is transformed into a DSR. The main idea is to *reverse engineer* the universal dependencies annotation guidelines[10].

3.1 Morphology

Word forms in UD are lists without children that are mapped to terminal symbols in jsRealB. So we transform the UD notation to the DSR one by mapping lemma and feature names. For example,

UD	jsRealB
`[noun:"mother",number:sing]`	`n("mother")*n("s")`
`[verb:"be",mood:ind,number:sing,person:3, tense:pres,verbform:fin]`	`v("be")*t("p")*pe(3)`
`[pron:"we",case:nom,number:plur,person:1, prontype:prs]`	`pro("I")*n("p")*pe(1)`

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[11], it is specified that

> *treebanks have considerable leeway in interpreting what "canonical or base form" means*

In the English UD corpora, it seems that the `LEMMA` of pronoun is always the same as its `FORM`. We decided to *lemmatize further* instead of merely copying the lemma as a string input to jsRealB so that verb agreement can be performed.

What should be a `LEMMA` is an hotly discussed 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 do a more `agressive` lemmatization to lower the number of base forms to with help further NLP processing that is often dependent on the number of different types.

3.2 UD to Deep Syntactic Representation

The essential idea is to transform recursively each child to produce a list of DSRs labeled with the name of the relation. The head of the relation is used as the constituent to which are added the dependents.

According to the annotation guidelines, there are two main types of dependents: nominals and clauses[12] which themselves can be simple or complex.

Nominals are triggered when the head is either a noun, an adjective, a proper noun, a pronoun or a number. When it is a noun, most often a `NP` is created using information

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

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

[12] not to be confused with the Prolog clauses...

gathered from the dependents depending on their part of speech tags such as `det`, `nummod`, `amod`, `compound` or `nmod:poss`. Special cases are needed for proper nouns, possessives with `'s`, prepositional phrases and appositions. Nouns and adjectives can be transformed to a sentence when its dependent is a `nsubj` with a possible `cop`; if the copula is not given, then `be` is used.

Clauses (both simple and complex) are triggered when a verb is encountered as the head. In this case, a `S` is created taking as subject a `expl` or `nsubj`; the `V` of the `VP` is the lemma of the head and the complements are all other dependencies in order of appearance which corresponds to the order of the original sentence.

Prepositional phrases are dealt specially by removing the preposition and dealing with the other dependents like an *ordinary* clause that is then nested into the prepositional phrase. Proper nouns with `flat` dependents are built beforehand.

This mechanism (25 rules in 100 lines of commented and indented Prolog) was first developed by reading the annotation guidelines and then refined by experience on the UD corpus.

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 before starting the transformation previously described, the structure is checked for the occurrence of `part:"not"` and an auxiliary to generate the `.typ({neg:true})` option for jsRealB; 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.

4 Adaptation of UD-SurfR for the tasks at SR'19

The data used by this shared task was created by modifying original UD structures to create two tracks:

Shallow Track (T1) (see the first line of Table 3 the transformed example shown in Figure 1) in which the word order (i.e. the lines) has been permuted and only lemmas have been kept; some information about linear order about the governor has been added. The relations between lines have been kept intact. The task thus consists in determining the word order and inflecting the words.

Deep Track (T2) (see the second line of Table 3 for a second transformation applied to T1) in which functional such as determiners (`DET`) or auxiliaries (`AUX`) and surface-oriented morphological information such as prepositions (`ADP`) have been removed from the T1 structures. The goal of T2 is to reintroduce the now missing functional words and morphological features.

The creation of the data set is described in [4]. The output of the systems has been evaluated using automated metrics and manually by means of Mechanical Turk.

T1	1	_	be	AUX	VBD	Mood=Ind...	9	aux:pass
	2	_	attack	NOUN	NN	Number=Sing|...	9	obl
	3	_	mother	NOUN	NN	Number=Sing|...	9	nsubj:pass
	4	_	he	PRON	PRP$	Gender=Masc|...	3	nmod:poss
	5	_	the	DET	DT	Definite=Def|...	2	det
	6	_	.	PUNCT	.	lin=+1|...	9	punct
	7	_	also	ADV	RB	original_id=4	9	advmod
	8	_	in	ADP	IN	original_id=6	2	case
	9	_	kill	VERB	VBN	Tense=Past|...	0	root
T2	1	_	kill	VERB	_	Tense=Past|id2=1|id1=9|original_id=5|...	0	ROOT
	2	_	mother	NOUN	_	Number=Sing|id1=3|original_id=2 ...	1	A2
	3	_	also	ADV	_	id1=7|original_id=4 ...	1	A1INV
	4	_	attack	NOUN	_	Number=Sing|id2=5|id1=2|id3=8|origina...	1	AM
	5	_	he	PRON	_	Number=Sing|id1=4|Poss=Yes|original_i...	2	AM

Table 3: UD structure of Table 1 as modified for the T1 and T2 tasks of SR'19

The organizers of SR'19 have taken for granted that these tasks would be solved using statistical and machine learning approaches, which is an *obvious* way given the recent trends in NLP. They provide a list of *authorized* resources such as language models and distributed representations of words.

We decided to try an alternative approach by adapting UD-SurfR based on a symbolic approach to see how it compares with machine learning systems. We are aware that *we are not following the rules* of SR'19 as we use jsRealB, a system that is not *authorized* by the competition, but we think this experiment is still interesting. It does not require any specialized hardware and huge amount of memory as is often the case by modern machine learning approaches. It has been developed using only a few hand selected examples. These results could be used as a baseline on which statistical systems could build. We have deliberately shirked from adding any statistical techniques on the output of UD-SurfR just to determine how far a symbolic approach can go. In a production setting, it would surely be better to combine statistical and symbolic systems.

Given the fact that T1 structures are a permutation of the lines of the original structure, we conjectured that, once the tree structure would be retrieved, the difference would be minimal after *sorting* the leaves at each level of the tree. For T2, we took the original realizer and *abstracted* the name of the dependencies by *reversing* the transformations described in [4]. We now describe these modifications in more detail.

4.1 T1 to Deep Syntactic Representation

The algorithm given in the previous section cannot be used directly on the input of the Shallow Track because the word order has been permuted while keeping the intact the relations between the words. It means that the left or right position information of a child with respect to the head cannot be used anymore.

Proper lemmatization is performed by jsRealB. Unfortunately, lemmatization for T1 is not always systematic, there are a few cases such as *grounds* or *rights* where the plural was left in the lemma; no pronoun is lemmatized, so we find *he*, *them*, *she*, *it* while a canonical pronoun should be used, jsRealB uses *I*. Not having to find the appropriate pronoun simplifies realization because this is one of the main difficulties of English whose morphology is otherwise relatively simple at least compared to other languages.

Given the fact that the permutation left intact the links between the words, we used a very simple approach: we first build the tree and then sort the dependents at each level. The sorting first takes into account the information about the linear order added to make sure proper nouns and punctuation can be added at the appropriate place. Then a fixed order of relation name is chosen so that a subject appears before the verb or its complements, a determiner will be placed before an adjective and a noun, etc.

Once the T1 structure has been sorted, it is processed like a UD structure using the algorithm described above. In this case, the algorithm does not use the fact the left or right position of the children in relation to the head; this relation being lost by the permutation applied in creating T1. For the two previous examples, the sorting process recreates almost exactly the structure of the original UD and the output sentence is the same. This happens because small differences in the placement of `aux`, `mark` or `prep` do not change the realization.

Comparing the automatic results on the train and development sets shown in Table 6, we see that the results for T1 are only slightly worse than the ones for UD (especially for BLEU), so we consider that this approach is valuable for this special task.

It seems to us that the permutation of the lines in the dependency file does not change the input so much to warrant a special task. In fact, from an NLG point of view, it is artificial, as we cannot imagine a generation system that determines all the tokens but in a random order.

4.2 T2 to Deep Syntactic Representation

The SR'19 documentation dataset[13] provides a mapping between the universal dependencies and the ones used for T2. So we adapted the algorithm given for the UD by changing the names of the relations. Initially we thought that it would be possible to use the same algorithm as for T1 by mapping the relation names. But because of many special cases it became too complicated, so we *simplified* the original algorithm to deal separately with the smaller set of relation names for T2.

Like previously, the tree is built by reading the T2 dependencies and the dependents are sorted at each level depending on the name of the relations. Then the `NAME` dependents are processed using the linear order information.

Using a similar process as described for UD, we deal with nominals and clauses. For nominals, all `A1` and `AM` dependents are used for building a `NP`. A sentence is built when a verb is encountered as a head, the subject being the value of the `A1` relation, the verb phrase comprises the head verb and all other dependents. Some care has to be given to the

[13] `http://taln.upf.edu/pages/msr2019-ws/srst_dataset_doc.txt`

`Ai INV` relations that are used as relative sentences for verbs and noun complements. Given the fact that important information has been removed in the T2 structures, the results *leave much room for improvement*. It would surely be interesting to improve this output using a statistical spell or style checker.

Table 4 shows the T2 dependency structure for the example of Table 1. The fact that this sentence should be written in passive mode is not given explicitly in the input, but the transformation rules indicate that a passive subject is indicated by a `A2` relation without any `A1`. Prepositions being absent from T2 structures, we computed the most frequent preposition used with each word as head in the original UD corpora (this is the only statistical process used in our system, but there should be more). This preposition is added for all dependents having relation `AM` and `Ai` (`i`>= 3). For the verb *kill*, the most frequent preposition being *in*, this is why it is added (correctly in this case) before *the attack*. Finally we see that the original sentence was reproduced *verbatim*, but this is not always the case...

```
['ROOT'>r,[verb:"kill",tense:past,clausetype:dec],
          ['A2'>r,[noun:"mother",number:sing],
                  ['AM'>r,[pron:"he",number:sing,poss:yes,person:3,
                     prontype:prs]]],
          ['A1INV'>r,[adv:"also"]],
          ['AM'>r,[noun:"attack",number:sing,definite:def]]]
```

```
s(vp(v("kill")*t("ps"),
     adv("also"),
     np(d("my")*pe(3)*ow("s")*n("s")*g("m"),
        n("mother")*n("s")),
     pp(p("in"),
        np(d("the"),
           n("attack"))*n("s"))))*typ({pas:true})
```

```
His mother was killed also in the attack.
```

Table 4: The T2 dependency given in the second line of Table 3 into the nested list structure shown in the first line; the `id` and `original_id` features are ignored as they are given for easing the training of learning algorithms. It is then transformed into a Deep Syntactic Representation shown in the second line and then into a Surface Syntactic Representation (not shown here) which is given to jsRealB to realize the sentence shown at the bottom which is the same as the original sentence given at the top of Table 1.

Table 5 shows the T2 tree structure, the Surface Syntactic Representation and the realized sentence for the example of Table 2. As the *expletive* has been transformed for this task into the verb *be* with subject *nothing*, the realized sentence starts awkwardly.

```
['ROOT',[verb:"be",tense:pres,clausetype:dec],
        ['A1',[pron:"nothing",number:sing],
              ['A2INV',[verb:"do",tense:pres,mood:pot],
                       ['A1',[pron:"we",number:plur,person:1,prontype:prs]],
                       ['A1INV',[adv:"really"]],
                       ['AM',[pron:"it",number:sing,person:3,prontype:prs]]]],
        ['A2INV',[cconj:"and"]],
        ['PARATAXIS',[verb:"live",tense:pres,aspect:prog],
                     ['A1',[noun:"people",number:plur],
                           ['A1INV',[verb:"suggest",tense:pres,aspect:prog],
                                    ['A2',[verb:"go",tense:pres],
                                          ['A1',[pron:"we",number:plur,person:1,
                                                 prontype:prs]],
                                          ['A1INV',[adv:"out"]],
                                          ['LIST',[verb:"fight",tense:pres],
                                                  ['A2',[pron:"they",number:plur,person:3,
                                                         prontype:prs]],
                                                  ['A2INV',[cconj:"and"]]]]]],
                     ['AM',[noun:"land",number:sing],
                           ['NAME',[noun:"dream",lin:(-1),number:sing]]]]]
```

```
sp(c("and"),
   s(ls(s(pro("I")*n("p")*pe(1)*g("m"),
          vp(v("do")*t("p"),
             adv("really"),
             pp(p("in"),
                pro("me")*n("s")*pe(3)*g("n"))))*typ({(mod):"poss"}),
        pro("nothing")*n("s")),
     vp(v("be")*t("p"),
        s(ls(s(vp(v("suggest")*t("p"),
                  s(pro("I")*n("p")*pe(1)*g("m"),
                    vp(v("go")*t("p"),
                       adv("out"),
                       sp(c("and"),
                          s(vp(v("fight")*t("p"),
                               pro("me")*n("p")*pe(3)*g("m")))*typ({pas:true})))))
                                                          *typ({pas:true,prog:true}),
             n("people")*n("p")),
          vp(v("live")*t("p"),
             pp(p("in"),
                np(n("dream")*n("s"),
                   n("land")*n("s")))))*typ({prog:true}))))
```

```
We can do really in it nothing is and is suggesting to we go out fights for
them and people is living in dream land.
```

Table 5: The first line shows the Deep Syntactic Representation corresponding to the T2 for the sentence of Table 2; it is then transformed into a Surface Syntactic Representation (not shown here) which is given to jsRealB to realize the sentence shown at the bottom which can be compared with the original sentence given at the top of Table 2.

5 Results and evaluation

We ran the program on all training (≈ 20 000 sentences) and development (≈ 4 000 sentences) sets provided by the organizers of SR'19 for English. We also ran them on the test sets of the 2018 and 2019 competitions. Using SWI-Prolog V8.1, the whole set is processed in about 5 minutes of real time (half of which is CPU) on a 2.2 GHz MacBook Pro, including the production of evaluation files (a good example of *Green AI* [8]).

5.1 Automatic evaluation

5.1.1 Development and Test sets of SR'19

Table 6 shows the BLEU[14], NIST[15] and DIST[16] scores on the training and development set for the four English corpora. The scores for T1 and UD are quite similar and their value are within the scores obtained by systems on a similar task in 2018.

	ewt			gum			lines			partut		
	BLEU	DIST	NIST	BLEU	DIST	NIST	BLEU	DIST	NIST	BLEU	DIST	NIST
train	12 543 sent.			2 914 sent.			2 738 sent.			1 781 sent.		
UD	0.49	0.64	12.26	0.48	0.58	10.93	0.46	0.56	10.53	0.44	0.48	10.37
T1	0.38	0.62	10.88	0.40	0.55	10.17	0.36	0.53	9.42	0.38	0.48	9.73
T2	0.25	0.56	9.24	0.26	0.47	8.63	0.24	0.49	8.11	0.24	0.42	8.03
dev	2002 sent.			707 sent.			912 sent			156 sent.		
UD	0.48	0.69	10.39	0.49	0.60	9.87	0.48	0.60	9.96	0.39	0.61	7.76
T1	0.37	0.66	9.33	0.41	0.57	9.23	0.38	0.56	9.00	0.33	0.60	7.31
T2	0.25	0.64	8.21	0.25	0.49	7.83	0.24	0.54	7.66	0.23	0.54	6.46

Table 6: Automatic evaluation scores produced by the evaluation scripts of the SR'19 organizers on the train and dev sets. For UD and T1, the scores seem competitive with the ones obtained by the participants at the 2018 competition shown in Table 7.

5.1.2 2018 Shared task

Table 7 gives these scores for the test set used in the 2018 competition. These scores are competitive for T1 and only slightly less in BLEU than the single 2018 participant to T2. A cursory manual evaluation of the output for T2 shows the need for improvement for long sentences even though the automatic scores are quite similar except for BLEU. This can be explained by the fact that a lot of information is not given in the output but is expected

[14]Precision metric that computes the geometric mean of the n-gram precisions between generated text and reference texts and adds a brevity penalty for shorter sentences.

[15]N-gram similarity metric weighted in favor of less frequent n-grams which are taken to be more informative.

[16]Normalized edit distance: inverse, normalized, character-based string-edit distance that starts by computing the minimum number of character inserts, deletes and substitutions(all at cost 1) required to turn the system output into the (single) reference text.

to be inferred by the NLG system. For the moment, the only *real* information added by the system is the most frequent preposition encountered in the test and development set for complements of nouns or of verbs.

	BLEU	DIST	NIST
T1	0.38	0.69	9.38
2018-best	0.69	0.80	12.02
2018-worst	0.08	0.47	7.71
T2	0.19	0.60	7.87
2018	0.22	0.49	6.95

Table 7: Automatic evaluation on the 2061 sentences of the 2018 test set compared with the scores obtained by systems participating in the 2018 shared task. Only one system provided output for the T2 task.

5.1.3 2019 Shared Task

The 2019 test set was much more comprehensive with more languages and different types of corpora. We submitted output for English only for which there were *regular* test files extracted from the original UD files. There was also an out of domain test file (prefixed with `pud) in Table 8` and test files using parser outputs from the 2018 UD parsing shared task (containing `Pred` in their names in Table 8).

Evaluation was done on both tokenized and *detokenized* input.As jsRealB was already realizing a *detokenized* output, we had to write a tokenizer to separate the tokens in order to make the output comparable with the one produced by other systems working at the token level and that applied some postprocessing to produce a more readable output with proper casing and appropriate spacing around punctuation.

Table 8 shows that, in terms of automatic scores, UD-SurfR is competitive: it is more or less the average between the best and worst scores obtained by other systems. And this seems consistent across the different types of input files: regular, out of domain or produced by parsers. The score for *detokenized* output (Table 9) are slightly worse than the tokenized input, but this is expected for automatic measures that operate on tokens that depend on the specifics of the tokenization process.

The score for *detokenized* output (Table 9) are slightly worse than the tokenized input, but this is expected for automatic measures that operate on tokens. The different types of input files, (regular, out of domain or produced by parsers)

	# sent.	BLEU			DIST			NIST		
		UDS	min	max	UDS	min	max	UDS	min	max
T1 English only										
ewt-ud-test	2 077	0.41	0.22	0.86	0.60	0.46	0.98	0.11	0.10	0.14
gum-ud-test	778	0.47	0.15	0.84	0.59	0.38	0.84	0.11	0.09	0.13
lines-ud-test	914	0.41	0.15	0.81	0.57	0.40	0.82	0.10	0.08	0.13
partut-ud-test	153	0.48	0.07	0.87	0.58	0.36	0.86	0.09	0.03	0.11
pud-ud-test	1 000	0.47	0.12	0.87	0.59	0.36	0.87	0.11	0.09	0.13
ewt-Pred-HIT-edit	1 785	0.40	0.21	0.82	0.59	0.44	0.85	0.10	0.10	0.13
pud-Pred-LATTICE	1 032	0.42	0.13	0.83	0.58	0.37	0.86	0.11	0.09	0.13
average		0.44	0.15	0.84	0.58	0.40	0.87	0.11	0.08	0.13
T2 English only										
ewt-ud-test	2 077	0.26	0.23	0.55	0.55	0.55	0.76	0.09	0.07	0.12
gum-ud-test	778	0.26	0.18	0.52	0.52	0.49	0.73	0.09	0.06	0.11
lines-ud-test	914	0.25	0.21	0.47	0.51	0.51	0.72	0.09	0.06	0.11
partut-ud-test	153	0.24	0.17	0.46	0.49	0.47	0.67	0.08	0.05	0.09
pud-ud-test	1 000	0.26	0.18	0.51	0.50	0.50	0.72	0.10	0.06	0.11
ewt-Pred-HIT-edit	1 785	0.25	0.22	0.54	0.53	0.53	0.75	0.09	0.07	0.12
pud-Pred-LATTICE	1 032	0.24	0.17	0.48	0.50	0.50	0.72	0.09	0.06	0.11
average		0.25	0.20	0.50	0.51	0.51	0.73	0.09	0.06	0.11

Table 8: Automatic evaluation of the *tokenized* output of UD-SurfR (UDS) on the 7 English corpora (7 739 sentences) of the 2019 test set compared with the minimum and maximum scores obtained by systems participating in the shared task. The horizontal lines in the tables separate regular files (first group), from an *out of domain* corpus (pud-ud-test) and the last two are UD produced by automatic parsers. 12 systems participated to the T1 task, and 4 to the T2 task.

	# sent.	BLEU			DIST			NIST		
		UDS	min	max	UDS	min	max	UDS	min	max
T1 English only										
ewt-ud-test	2 077	0.37	0.37	0.75	0.60	0.60	0.86	0.09	0.09	0.13
gum-ud-test	778	0.42	0.40	0.79	0.59	0.56	0.83	0.09	0.08	0.12
lines-ud-test	914	0.37	0.27	0.76	0.57	0.53	0.82	0.09	0.08	0.12
partut-ud-test	153	0.44	0.05	0.83	0.58	0.51	0.86	0.08	0.03	0.11
pud-ud-test	1 000	0.43	0.41	0.82	0.60	0.54	0.87	0.10	0.10	0.13
ewt-Pred-HIT-edit	1 785	0.36	0.00	0.76	0.59	0.00	0.85	0.09	0.00	0.13
pud-Pred-LATTICE	1 032	0.37	0.37	0.78	0.58	0.55	0.86	0.09	0.09	0.13
average		0.39	0.27	0.78	0.58	0.47	0.85	0.09	0.07	0.12
T2 English only										
ewt-ud-test	2 077	0.24	0.20	0.49	0.55	0.54	0.76	0.08	0.07	0.11
gum-ud-test	778	0.24	0.15	0.49	0.51	0.49	0.73	0.08	0.05	0.10
lines-ud-test	914	0.22	0.18	0.43	0.51	0.51	0.72	0.07	0.06	0.10
partut-ud-test	153	0.21	0.14	0.42	0.48	0.47	0.67	0.07	0.04	0.08
pud-ud-test	1 000	0.24	0.16	0.48	0.50	0.50	0.73	0.08	0.06	0.11
ewt-Pred-HIT-edit	1 785	0.22	0.19	0.49	0.52	0.52	0.75	0.08	0.06	0.11
pud-Pred-LATTICE	1 032	0.22	0.15	0.45	0.50	0.50	0.72	0.08	0.06	0.10
average		0.23	0.17	0.46	0.51	0.50	0.73	0.08	0.06	0.10

Table 9: Automatic evaluation of the *detokenized* output of UD-SurfR (UDS) on the 7 English corpora (7 739 sentences) of the 2019 test set compared with the minimum and maximum scores obtained by systems participating in the shared task. The horizontal lines in the tables separate regular files (first group), from an *out of domain* corpus (pud-ud-test) and the last two are UD produced by automatic parsers. 8 systems produced tokenized output for the T1 task, and 4 for the T2 task.

Figure 2 shows a graph of the BLEU scores for the tokenized sentences which seem to be typical of the comparison across the scores for all participants to the tasks. For T1 (left part of the figure), the score for UD-SurfR (the black stripped bar on the left) is approximately in the middle of the score of other systems. For T2, except for the very best system, UD-SurfR does surprisingly well compared with other participants.

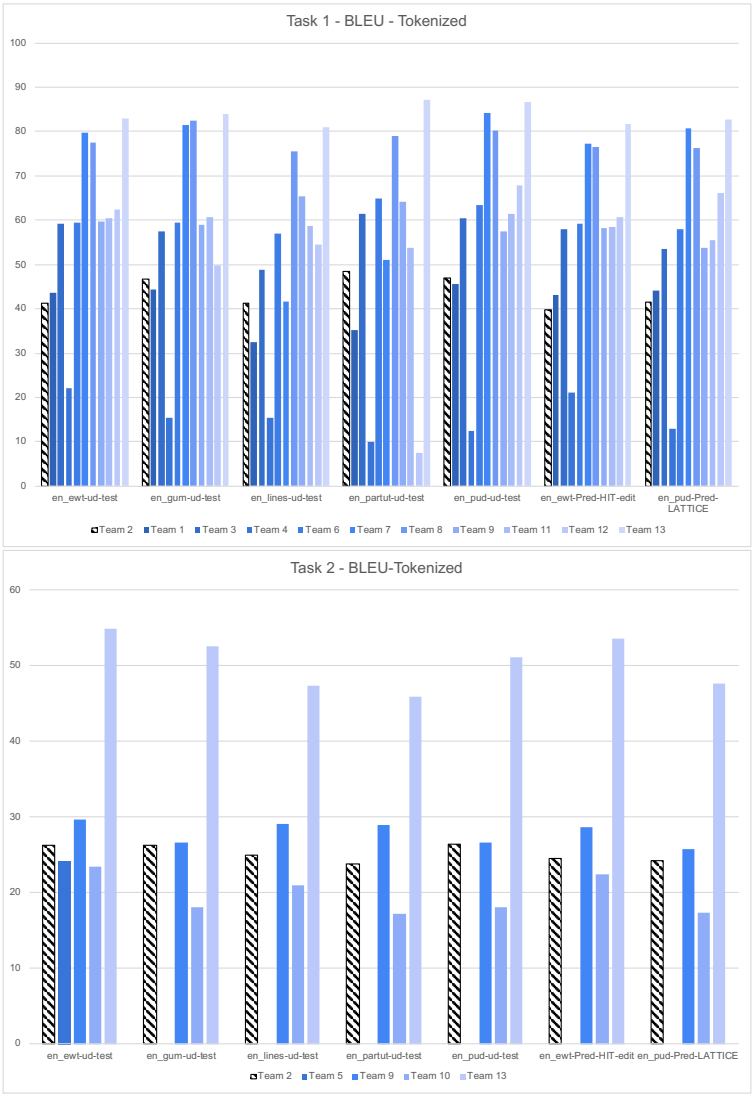


Figure 2: Comparison for BLEU scores for T1 (left) and T2 (right) on tokenized sentences from the English corpora for UD-SurfR (Team 2) shown as the black stripped bar to the left compared with the scores obtained by other participants.

5.2 Human Evaluation

The SR'19 organizers designed a human evaluation[17] a small subset (around 1000 sentences selected between all types of corpora) of the output of 12 systems and our 10-line baseline on two aspects:

- *The text adequately expresses the meaning of the sentence* for which our T1 system obtained 73% (ranked 11th) and T2 obtained 68% (ranked 14th) after scoring about 700 sentences; in fact these scores are not statistically different from each other.
- *the text reads well and is free from grammatical errors and awkward constructions* for with our T1 system obtained 58% which corresponds to the second group of system over 4. Note that the human reference only obtained 71% on this evaluation. These scores were based on about 550 sentences. We were quite surprised to see that the T2 system managed to get 50% even though no effort was put in adding any language model.

Given the relative simplicity of our approach, we are quite satisfied with these scores.

	T1			T2		
	av-BLEU	Gramm.	Meaning	av-BLEU	Gramm.	Meaning
Reference	100.0	71.1		100.0	71.1	
ADAPT	62.2	68.2	86.6			
BME-UW	49.9	58.3	73.5			
CLaC	13.7	48.1	60.9			
CMU	68.4	62.4	82.5			
DepDist	49,8	60.5	79.3			
DipInfo-UniTo	36.0	49.6	69.5			
IMS	73.3	67.9	85.6	50.4	61.9	80.6
LORIA	51.1	62.5	77.0			
RALI	38.2	57.5	72.9	25.2	50.3	68.3
Surfers				19.6	60.8	67.0
OSU-FB	46.1	57.4	78.4			
TilBurg	52.6	59.2	79,7			
10line base	7.3	36.5	55.3	1.0	37.8	53.0

Table 10: Result of the manual evaluation on a subset of the *detokenized* output of the systems submitting to the English part of the SR'19 competition for T1 and T2. The first column in each part, shows the average BLEU score on the corpora with the score obtained for the grammaticality of the output and the preservation of the meaning. The last line shows the scores obtained by the 10 line baseline system shown in Section 9. Figure 3 shows a graphical view of this data.

[17]The evaluation was done using the *Mechanical Turk* infrastructure

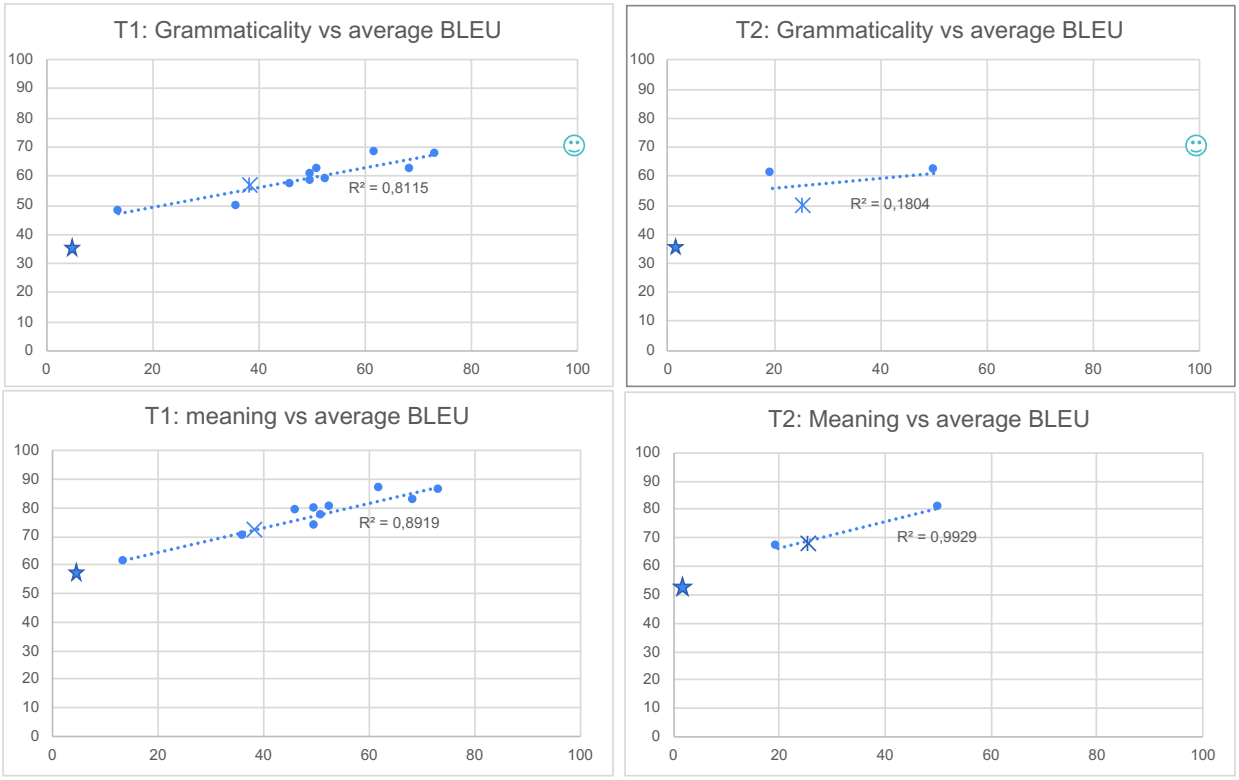


Figure 3: Comparison between BLEU scores and human evaluation for some T1 (left) and T2 (right) for both grammaticality and meaning preservation. The star shows the score of the 10-line baseline and the *smiley* shows the score of the reference for the grammaticality score. The score of UD-SurfR is shown with a star.

6 Conclusion

We have described a symbolic approach for tackling the tasks T1 and T2 of SR'19. We first described the development of UD-SurfR, a text realizer for standard UD input that can be used for checking the annotation. We then described UD-SurfR was modified to take into account the specificities of the shared task. The system has processed the training, development and test sets of the competition and obtained average results compared to other machine learning approaches. This is quite surprising given the fact, that the symbolic system only used a very small part of the training and development corpora. But more important, the experiment has revealed that task T1 (for English at least) is perhaps too easy and does not really correspond to a realistic input for a text realizer. T2 proved more challenging but the results produced by UD-SurfR proved to be similar to that of the few other competitors.

Acknowledgements

We thank Philippe Langlais who made detailed suggestions for improving the organization of the paper. We also thank Fabrizio Gotti for many fruitful discussions and suggestions.

References

[1] 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.

[2] 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.

[3] 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.

[4] Simon Mille, Anja Belz, Bernd Bohnet, and Leo Wanner. Underspecified universal dependency structures as inputs for multilingual surface realisation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 199–209, Tilburg University, The Netherlands, November 2018. Association for Computational Linguistics.

[5] 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.

[6] 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).

[7] Aarne Ranta and Prasanth Kolachina. From universal dependencies to abstract syntax. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 107–116, Gothenburg, Sweden, May 2017. Association for Computational Linguistics.

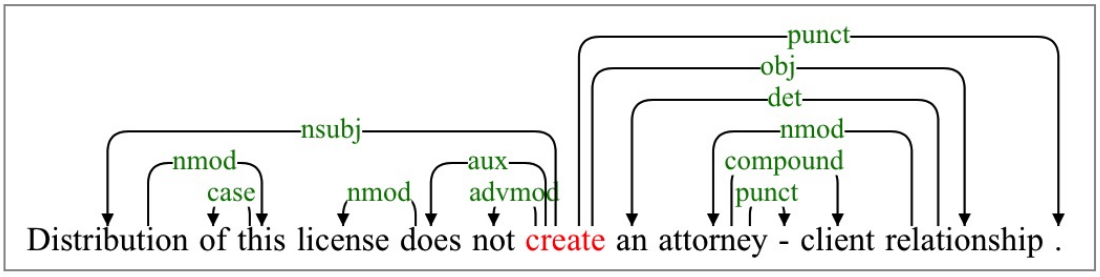
[8] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. arXiv-1907.10597, 2019.

Appendix

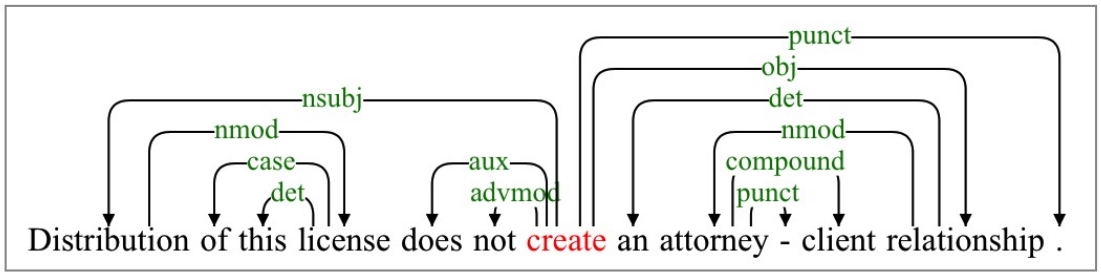
7 Annotation Error

The first UD in the file `en_partut-ud-train.conllu` corresponding to the sentence *Distribution of this license does not create an attorney-client relationship.* was realized by UD-SurfR as *Distribution this of doesn't create doesn't license a relationship client attorney.* which seemed awkward.

But a closer examination revealed that the original UD annotation was erroneous, because it corresponds to the following graph:



In which there is no link between *license* and *this*, but there is a link between *distribution* and *this* instead. This explains the realization produced by UD-SurfR. After correcting the annotation by changing links and a relation name between the first four words, the following graph is obtained:



UD-SurfR then realizes *Distribution of this license doesn't create a relationship client attorney.* which is still not perfect, but at least the discrepancy does not depend on what we consider to be an incorrect annotation.

This *error* was later confirmed by the maintainer of the PARTUT corpus. In fact, the error had already been corrected in a subsequent version (V2.4) of the corpus, but version 2.3 was the one used for SR'19.

8 Non-projective Dependencies

In our corpora, the dependencies of about 5% of the sentences are non-projective, which means that all words spanned by a head are not contiguous in the sentence. We are under the impression that most of the time, these types of dependencies are a symptom of annotation errors, but there can be legitimate cases, see Figure 4.

We show here a few examples.

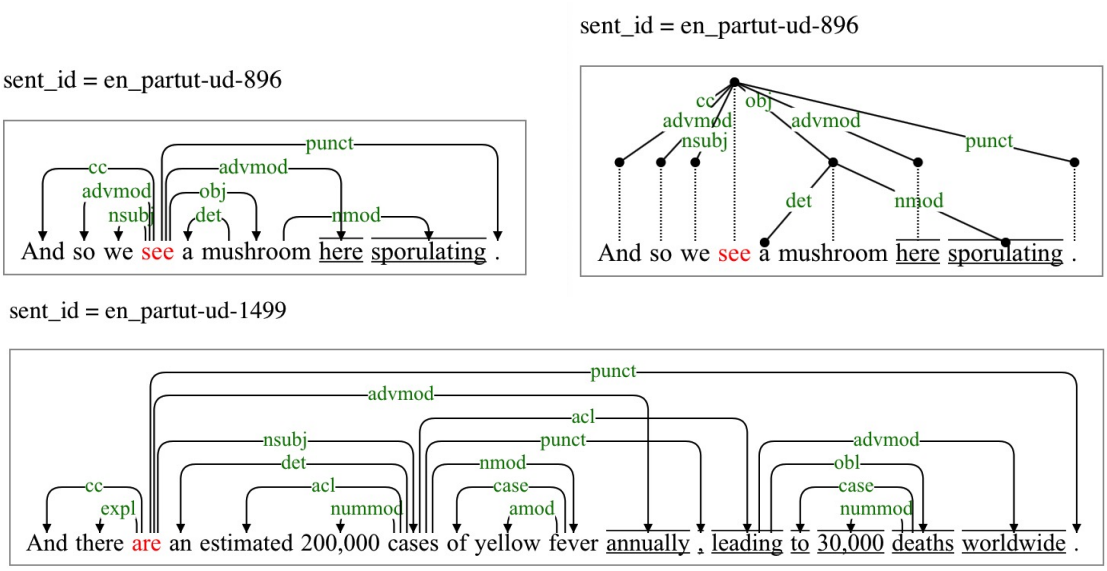


Figure 4: Two examples of legitimate non-projective dependencies found in the Partut corpus. The non-contiguous words in a span are highlighted with under and over lines.

9 Appendix - Baseline generator

The following listing shows the 10 lines of Python code for a baseline generator from a UD file using the `pyconll` API that allows the creation of a tree from a UD format. The generator performs an in-order traversal of the tree to produce a list of tokens (i.e. the `FORM` is it is given, otherwise the `LEMMA`).

```
import pyconll

#### baseline generator from UD file with an in-order traversal of the tree read by pyconll
##    and outputing the form at each node
#    some care must be taken to ignore non numeric id (e.g. 10-11)

##  Unfortunately pyconll does not keep separate lists for left and right children
##  it seems that all left children appear before the right ones,
##  but we do not take it for granted, so we compare the id of the child with the one of the
     head
def isLeftChild(headId,childId):
    return childId.isnumeric and int(childId)<headId

def gen(tree):
    treeId=int(tree.data.id)
    ## HACK: sum(list of lists,[]) concatenates all sublists to a single list
    return (sum([gen(child) for child in tree if isLeftChild(treeId, child.data.id)],[])
            + [tree.data.form if tree.data.form != "_" else tree.data.lemma] + ## use lemma
                if form is missing
            sum([gen(child) for child in tree if not isLeftChild(treeId, child.data.id)],[])
                )

def testSingleFile(fileName):
    for sentence in pyconll.load.iter_from_file(fileName):
        print("␣".join(gen(sentence.to_tree())))
```

For completeness, we show the scores obtained by this baseline on both the 2019 training and development set (Table 11) and on the SR'19 (Table 12). Table 13 shows the results on the test sets for 2019. As expected, the results are excellent for UD, but catastrophic for T1 and T2 for which the lines of the dependents have been permuted and the forms replaced by their lemmas.

	ewt			gum			lines			partut		
	BLEU	DIST	NIST	BLEU	DIST	NIST	BLEU	DIST	NIST	BLEU	DIST	NIST
train	12 543 sent.			2 914 sent.			2 738 sent.			1 781 sent.		
UD	0.65	0.92	12.38	0.63	0.92	11.01	0.61	0.92	10.73	0.65	0.92	10.87
T1	0.05	0.31	6.72	0.05	0.22	6.55	0.04	0.19	5.72	0.05	0.17	6.20
T2	0.01	0.29	5.45	0.01	0.20	5.17	0.01	0.18	4.27	0.01	0.15	4.42
dev	2002 sent.			707 sent.			912 sent			156 sent.		
UD	0.63	0.89	10.23	0.64	0.95	9.81	0.61	0.94	9.82	0.60	0.92	7.75
T1	0.05	0.38	6.31	0.05	0.22	6.10	0.04	0.18	5.60	0.04	0.14	5.03
T2	0.01	0.36	5.38	0.01	0.21	4.63	0.01	0.17	4.20	0.01	0.15	3.87

Table 11: Automatic evaluation scores of the baseline generator produced by the evaluation scripts of the SR'19 organizers on the train and dev sets. For UD the scores are excellent which is not surprising but as expected they are catastrophic for T1 and T2.

	BLEU	DIST	NIST
T1	0.05	0.36	6.42
2018-best	0.69	0.80	12.02
2018-worst	0.08	0.47	7.71
T2	0.01	0.37	5.53
2018	0.22	0.49	6.95

Table 12: Automatic evaluation for the baseline generator on the 2061 sentences of the 2018 test set compared with the scores obtained by systems participating in the 2018 shared task. Only one system provided output for the T2 task.

		BLEU			DIST			NIST		
	# sent.	BSL	min	max	BSL	min	max	BSL	min	max
T1 English only										
ewt-ud-test	2 077	0.08	0.22	0.86	0.38	0.46	0.98	0.08	0.10	0.14
gum-ud-test	778	0.08	0.15	0.84	0.26	0.38	0.84	0.08	0.09	0.13
lines-ud-test	914	0.06	0.15	0.81	0.18	0.40	0.82	0.07	0.08	0.13
partut-ud-test	153	0.08	0.07	0.87	0.15	0.36	0.86	0.07	0.03	0.11
pud-ud-test	1 000	0.07	0.12	0.87	0.14	0.36	0.87	0.08	0.09	0.13
ewt-Pred-HIT-edit	1 785	0.07	0.21	0.82	0.29	0.44	0.85	0.08	0.10	0.13
pud-Pred-LATTICE	1 032	0.07	0.13	0.83	0.15	0.37	0.86	0.08	0.09	0.13
average		0.07	0.15	0.84	0.22	0.40	0.87	0.08	0.08	0.13
T2 English only										
ewt-ud-test	2 077	0.01	0.23	0.55	0.35	0.55	0.76	0.04	0.07	0.12
gum-ud-test	778	0.01	0.18	0.52	0.22	0.49	0.73	0.04	0.06	0.11
lines-ud-test	914	0.01	0.21	0.47	0.16	0.51	0.72	0.03	0.06	0.11
partut-ud-test	153	0.01	0.17	0.46	0.16	0.47	0.67	0.02	0.05	0.09
pud-ud-test	1 000	0.01	0.18	0.51	0.13	0.50	0.72	0.03	0.06	0.11
ewt-Pred-HIT-edit	1 785	0.01	0.22	0.54	0.23	0.53	0.75	0.05	0.07	0.12
pud-Pred-LATTICE	1 032	0.01	0.17	0.48	0.14	0.50	0.72	0.03	0.06	0.11
average		0.01	0.20	0.50	0.20	0.51	0.73	0.03	0.06	0.11

Table 13: Automatic evaluation of the *tokenized* output of the baseline generator (BSL) on the 7 English corpora (7 739 sentences) of the 2019 test set compared with the minimum and maximum scores obtained by systems participating in the shared task (this baseline generator was not included). The horizontal lines in the tables separate regular files (first group), from an *out of domain* corpus (pud-ud-test) and the last two are UD produced by automatic parsers. 12 systems participated to the T1 task, and 4 to the T2 task. **This is the same table as Table 8, except for the BSL columns**.