Realizing Universal Dependencies Structures using a symbolic approach

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

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.

1 Introduction

This paper describe the system that we submitted to the Surface Realization Shared Task 2019 (SR'19)[1] in conjunction with Second Workshop on Multilingual Surface Realization (Mille et al., 2019). The data used by this shared task was created by modifying original Universal Dependencies structures (Nivre et al., 2016) (UD) to create two tracks:

Shallow Track (T1) in which word order is permuted and tokens have been lemmatized and some information about linear order about the governor has been added. The task consists in determining the word order and inflecting the words.

Deep Track (T2) in which functional and surface-oriented morphological information has been removed from the T1 structures. The goal is to reintroduce the missing functional words and morphological features.

The creation of the data set is described in (Mille et al., 2018). The output of the systems have been evaluated using automated metrics and a subset of those, evaluated manually.

The organizers of SR'19 have taken for granted that this task would be solved using statistical and machine learning approaches, which seems to be an *obvious* way of going 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 first building UD-SURFR (Universal Dependency Surface Realizer), a symbolic system for the original UD structures and then adapting it for the two tasks of SR'19. We thought it would be interesting to see how this *classical* approach compares with machine learning systems.

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,[2] 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. The UD structures are provided in tab separated files in a well-defined format (see row 1 of Table 1 for a small example). 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 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

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

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

Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019), pages 41–49
Hong Kong, China, November 3rd, 2019.
https://doi.org/10.18653/v1/P17

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.

We did not find any text realizer that takes UD annotations as input except for Ranta and Kolachina (2017) 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.

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 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. During our development, we encountered a concrete example where the automatic realization revealed an error in the original annotation; the error was later confirmed by the maintainer of the corpus.

The next section shows the tree representations used by our system using a simple example from the training test. Section 3 describes the development of UD-SURFR for the original UD structures and how it was adapted for SR'19. As T1 structures are a permutation of the lines of the original structure, but we conjectured that, once the tree structure would be retrieved, the differences would be minor after *sorting* the leaves at each level of the tree. For T2, we took the realizer for T1 and *abstracted* the name of the dependencies by *reversing* the transformations described in (Mille et al., 2018). Section 4 gives the results of the evaluation obtained using the evaluation scripts provided with the task. We also compare our results with the automatic and manual scores obtained by other systems that participated in the task. We conclude with some lessons learned from this development.

2 Representations

Table 1 illustrates the transformation steps between an input UD (row 1) and an English sentence using a short example from the *train* set (`en_ewt-ud-train.conllu`). Row 1 shows the CONLLU format,[3] a series of tab separated lines into fields giving information for each token of the sentence. Row 2 shows the dependencies either as a set of links between words (on the left part) or as a tree (on the right) Row 3 shows the Prolog structure corresponding to the tree which is transformed into the Deep Syntactic Representation shown in Row 4. Row 5 shows the Surface Syntactic Representation which is used by JSREALB to realize the sentence shown in Row 6.

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: its HEAD (field 7) is 0. Its children are found by looking for lines that have the root as HEAD. Each child is then taken as the root of the subtree and recursively parsed and transformed in the following format, using the official CONLLU field names:

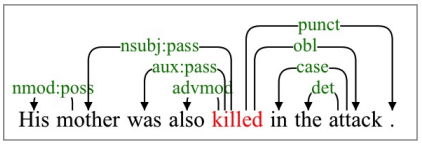
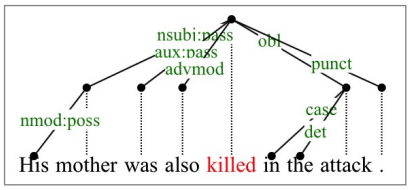
```
[DEPREL>LR, [UPOS:LEMMA | FEATS]
    | children]
```

LR is either `l` or `r` depending on whether 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 the value of its HEAD. This

[3] https://universaldependencies.org/format.html

1	Universal Dependencies in CONLLU	`# 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`

2	Linked and tree representations	 

3	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"]]]`

4	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(".")

5	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(".")

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

Table 1: Representations used in the transformation of the Universal Dependencies in CONLLU format in row 1 to the sentence shown in row 6.

is useful in some cases for putting compounds and complements before or after the head. But for the T1 and T2, this information is not reliable because the nodes have been permuted and it is the job of the realizer to get the compound and complements in the right order.

2.2 Deep Syntactic Representation (DSR)

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 Surface Syntactic Representation (SSR) described in the next subsection. In principle, it would have been pos-

sible to create the SSR directly, but, for technical reasons, it proved more convenient to use this intermediary step.

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 (SSR)

The SSR is the input form 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 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"),
        N("apple"))).t("ps"))
```

is the JSREALB specification for `Women ate the red apple.` Plural is indicated with the option `n("p")` where `n` indicates the number and `"p"` plural; this explains why the determiner `"a"` does not appear in the output. 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.[4] It takes care of morphology, declension and conjugation to create well-formed texts. Some options allow adding HTML tags to the realized text.

An interesting feature of JSREALB, inspired by a similar mechanism in SIMPLENLG, is the fact that once the sentence structure has been built, many variations can be obtained by adding a set of options to the sentences, to get negative, progressive, passive, modality and some type of questions. 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.`, a negative passive sentence with a modal verb for possibility.

Row 5 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 has been 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.

3 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[5].

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, see Table 2.

As shown in the last example, we had to *normalize* pronouns to what JSREALB considers as its base form. In the morphology principles of UD[6], it is specified that

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

In 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 a 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

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

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

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

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)

Table 2: Some examples of mapping between UD features and JSREALB options

it would be preferable to do a more `aggressive` lemmatization to lower the number of base forms in order to 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[7] 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 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.

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

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.

3.3 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.

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 difficulties of English generation 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

```
# sent_id = weblog-juancole.com_juancole_20051126063000_ENG_20051126_063000-0020
# text = His mother was also killed in the attack.
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
```

```
['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 3: The T2 dependency given at the top is parsed into the nested list structure shown in the second 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 third 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.

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.

3.4 T2 to Deep Syntactic Representation

The SR'19 documentation dataset[8] 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.

The tree is built by reading the T2 dependencies and the dependents are sorted at each level according to the relation names. 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 `AiINV` 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 3 shows the T2 structure for the example shown in Table 1. That this sentence should be written in passive mode is not specified 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 pro-

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

cess 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*, it is added (correctly in this case) before *the attack*. The original sentence was thus reproduced *verbatim* but, of course, this is not always the case...

4 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* (Schwartz et al., 2019).

4.1 Automatic Evaluation

4.1.1 Training and development sets

Table 4 shows the BLEU, NIST and DIST scores on the 2019 training and development sets 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. Comparing the automatic results on the train and development sets, 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, T1 seems artificial, as we cannot imagine a generation system that determines all the tokens but in a random order.

4.1.2 Test sets

Table 5 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 unique participant for this task in 2018. 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 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.

The 2019 test set was much more comprehensive with more languages and different types of corpora. Evaluation was done on both tokenized and *detokenized* input. In the case of UD-SURFR, 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 of other systems working at the token level and then applying some postprocessing to produce a more readable output with proper casing and appropriate spacing around punctuation. 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 all types of corpora. Figure 1 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.

4.2 Manual Evaluation

The SR'19 organizers evaluated the output of 16 systems 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.

	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 4: 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 5.

	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 5: 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.

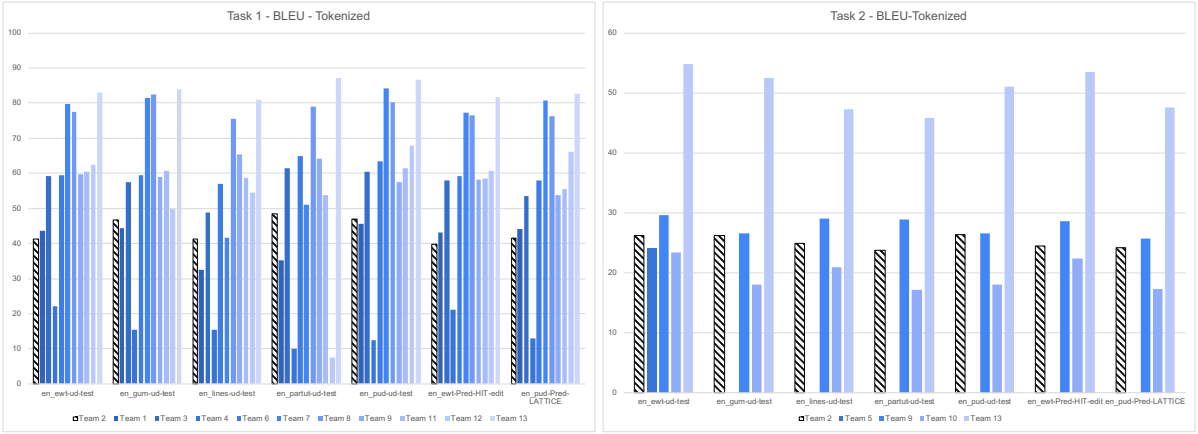


Figure 1: Comparison for BLEU scores for T1 (left) and T2 (right) on tokenized sentences from the English corpora for UD-SURFR (Team 2) shown as the first black stripped bar to the left compared with the scores obtained by other participants.

5 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 are finally relatively similar.

References

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. Association for Computational Linguistics.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, and Leo Wanner. 2019. 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.

Simon Mille, Anja Belz, Bernd Bohnet, and Leo Wanner. 2018. 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. 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. Association for Computational Linguistics.

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. 2016. 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. European Language Resources Association (ELRA).

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

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