Multilingual Back-and-Forth Conversion between Content and Function Head for Easy Dependency Parsing

Ryosuke Kohita **Hiroshi Noji** **Yuji Matsumoto**
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{kohita.ryosuke.kj9, noji, matsu}@is.naist.jp

Abstract

Universal Dependencies (UD) is becoming a standard annotation scheme cross-linguistically, but it is argued that this scheme centering on content words is harder to parse than the conventional one centering on function words. To improve the parsability of UD, we propose a back-and-forth conversion algorithm, in which we preprocess the training treebank to increase parsability, and reconvert the parser outputs to follow the UD scheme as a postprocess. We show that this technique consistently improves LAS across languages even with a state-of-the-art parser, in particular on core dependency arcs such as nominal modifier. We also provide an in-depth analysis to understand why our method increases parsability. [1]

1 Introduction

As shown in Figure 1 there are several variations in annotations of dependencies. A famous example is a head choice in a prepositional phrase (e.g, *to a bar*), which diverges in the two trees. Though various annotation schemes have been proposed so far (Hajic et al., 2001; Johansson and Nugues, 2007; de Marneffe and Manning, 2008; McDonald et al., 2013), recently the Universal Dependencies (UD) (de Marneffe et al., 2014) gains much popularity and is becoming the annotation standard across languages. The upper tree in Figure 1 is annotated in UD.

Practically, however, UD may not be the optimal choice. In UD a content word consistently dominates a function word, but past work points out that this makes some parser decisions more difficult than the conventional style centering on function words, e.g., the tree in the lower part of Figure 1 (Schwartz et al., 2012; Ivanova et al., 2013).

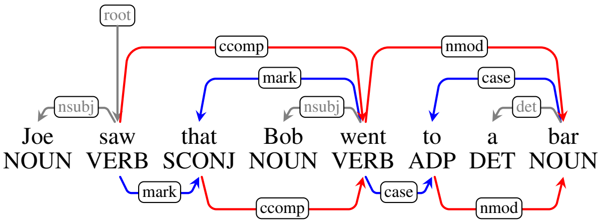

Figure 1: Dependency trees with content head (above) and function head (below).

To overcome this issue, in this paper, we show the effectiveness of a back-and-forth conversion approach where we train a model and parse sentences in an anontation format with higher parsability, and then reconvert the parser output into the UD scheme. Figure 1 shows an example of our conversion. We use the function head trees (below) as an intermediate representation.

This is not the first attempt to improve dependency parsing accuracy with tree conversions. The positive result is reported in Nilsson et al. (2006) using the Prague Dependency Treebank. For the conversion of content and function head in UD, however, the effect is still inconclusive. Using English UD data, Silveira and Manning (2015) report the negative result, which they argue is due to error propagation at backward conversions, in particular in copula constructions that often incur drastic changes of the structure. Rosa (2015) report the advantage of funcion head in the adposition construction, but the data is HamleDT (Zeman et al., 2012) rather than UD and the conversion target is conversely too restrictive.

Our main contribution is to show that the back-and-forth conversion can bring consistent accuracy improvements across languages in UD, by

[1] Our conversion script is available at https://github.com/kohilin/MultiBFConv

POS	Label	Example
ADP	case	... *a post* **about** *fault* ...
	dep	(ja) *Taro ni* **ha** ...
	mark	... *opinions* **on** *how it* ...
SCONJ	mark	*I think* **that** ...
ADV	mark	... *feet* **when** *you* ...
PART	case	*Elena* **'s** *motor cycle* ...
	mark	... *Sharon* **to** *make* ...

Table 1: The set of conversion targets. (ja) is an example in Japanese.

limiting the conversion targets to simpler ones around function words while covering many linguistic phenomena. Another limitation in previous work is the parsers: MSTParser or MaltParser is often used, but they are not state-of-the-art today. We complement this by showing the effectiveness of our approach even with a modern parser with rich features. We also provide an in-dpeth analysis to explore when and why our conversion brings higher parsability than the orignal UD.

2 Conversion method

Let us define notations first. For the i-th word w_i in a sentence, p_i denotes its POS tag, h_i the head index, l_i the dependency label, and *left*$_i$ (*right*$_i$) the list of indexes of left (right) children for w_i. For instance in the upper tree in Figure 1, w_5 = *went*, p_5 = VERB, h_5 = 2, l_5 = ccomp, and *left*$_5$ = [3, 4].

Forward Conversion The forward algorithm receives the original UD tree and converts it to a function head tree by modifying h_i. Figure 1 is an example, and Algorithm 1 is the pseudo-code; root(y) returns the root word index of tree y.

The algorithm traverses the tree in a top-down fashion and modifies the deepest node first. The modifications such as changing the mark arc from *went* to *that* in Figure 1 occur when it detects a word w_i (*that*, in this case), for which the pair (p_i, l_i) exists in the set of conversion targets, which is listed in Table 1 and is denoted by T in Algorithm 1. Let w_j be the head of the detected word w_i. Then, we reattach the arcs so that w_i's head becomes w_j's head and w_j's new head becomes w_i. Note that we modify heads (h_i) only and keep labels (l_i). We skip the children of the root word (line 13); otherwise, an arc with root label will appear at an intermediate node. We operate only on the outermost child when multiple candidates are found (line 11).

Backward Conversion In contrast, the backward algorithm receives a function head tree and reconverts it to a UD-style tree. Algorithm 2 is the pseudo-code.

Algorithm 1 Forward conversion

```
    Input: a dependency tree $y$ and the set of targets $T$.
    Output: modified $y$ after applying CONV(root($y$)).
 1: procedure CONV($j$)
 2:     for $i$ in left$_j$ do
 3:         CONV($i$)
 4:     CHANGEDEP(SEARCH(left$_j$), $j$)
 5:     for $i$ in right$_j$ do
 6:         CONV($i$)
 7:     CHANGEDEP(SEARCH(reverse(right$_j$)), $j$)
 8: procedure SEARCH(children)
 9:     for $i$ in children do
10:         if $(p_i, l_i) \in T$ then        ▷ $T$ is the set of targets.
11:             return $i$        ▷ The first found candidate is
    outermost. We only change this.
12: procedure CHANGEDEP($i, j$)
13:     if $l_j \neq$ root then        ▷ We skip the root.
14:         $h_i \leftarrow h_j$; $h_j \leftarrow i$
```

There are two main differences between the forward and backward algorithms. The first is the relative position of a target node (one of Table 1) among the operated nodes; in the forward algorithm they are the target node, its parent (head), and its grandparent, while in the backward algorithm they are the target node, its head, and its children. The second is how we reattach the nodes at the CHANGEDEP operation, in particular when the target node has multiple children. While the forward algorithm modifies only two arcs at once, the backward algorithm may modify more than two arcs considering possible parse errors at prediction. Specifically, when we find a target node having multiple children, we change the head of all these children to the head of the target (excluding those with the mwe label)[2]. We choose the intermost child as the new head of the target word (line 17).

Remarks The target list in Table 1 is developed for covering main constructions in English and Japanese while keeping the backward conversion accuracy high. We do not argue this list is perfect, and seeking better one is an important future work. Note also that we use this list across all languages.

One possible drawback of our method is that it may introduce additional non-projective arcs. In fact, we found that the ratio of non-projective arcs in the training sets increases by 10% points on av-

[2] In the original UD, tokens with mwe label sometimes attach to a function word, which may be the current target. To avoid flipping the relationship of mwe components, our backward algorithm skips them in the CHANGEDEP operation.

Algorithm 2 Backward conversion

```
Input: a dependency tree $y$ and the set of targets $T$.
Output: reconverted $y$ after applying CONV(root($y$)).
procedure CONV($j$)
    for $i$ in $left_j$ do
        CONV($i$)
    if $(p_j, l_j) \in T$ then
        CHANGEDEP($left_j$, $j$)
    for $i$ in $right_j$ do
        CONV($i$)
    if $(p_j, l_j) \in T$ then
        CHANGEDEP(reverse($right_j$), $j$)
procedure CHANGEDEP($children$, $j$)
    $lastchild \leftarrow -1$                        ▷ -1 is dummy.
    for $i$ in $children$ do
        if $l_i \neq$ mwe then                       ▷ We skip mwes.
            $lastchild \leftarrow i$
            $h_i \leftarrow h_j$
    if $lastchild \neq -1$ then
        $h_j \leftarrow lastchild$          ▷ The last chid is innermost.
```

erage. We argue this is not a serious restriction since UD already contains moderate amount of non-projective arcs and the parser should be able to handle them. In practice, this complication does not lead to performance degradation; when we employ non-projective parsers, the scores increase regardless of the increased non-projectivity.

3 Experiment

3.1 Experimental Setting

For each treebank and parser, we train two different models: one with the original trees (UD) and another with the converted trees (CONV). Reconverting CONV's output into the UD scheme by the backward algorithm, we can evaluate the outputs of both models against the same UD test set.

For parsers, we use two non-projective parsers: second-order MSTParser (MST) (McDonald et al., 2005) [3] and RBGParser (RBG) (Lei et al., 2014) [4] with the default settings, which utilizes the third-order features and is much stronger .

We choose 19 langueges from UD ver.1.3 considering the sizes and typological variations.[5] The ratio of converted tokens is 6.3% on average (2.3%-15.6%). The failed backward conversions rarely occur at most 0.01% (0.002% on average) in the training data. We use gold POS tags, and exclude punctuations from evaluation.

3.2 Result

Attachment scores Table 2 shows the main result and we can see that the improvements are remarkable in the labeled attachment score (LAS): For MST, the scores increase more than 1.0 point in many languages (11 out of 19), and for RBG, though the changes are smaller, more than 0.5 points improvements are still observed in 10 languages. The differences in the unlabelled attachment score (UAS) are modest, implying that our conversion contributes in particular to find correct arc *labels* rather than head words themselves. On the other hand, LAS of Hindi decreases with RBG. One possible explanation for this is that the score of original UD is sufficiently high (91.74) and our conversion may impede parsability in such cases.

These overall improvements are not observed in past work (Silveira and Manning, 2015). One reason of our success seems that we restrict our conversion to simpler constructions and operations. We do not modify copula and auxiliary constructions, which involve more complex changes, amplifying error propagation in backward conversion. Our conversion also suffers from such propagation (see below) but in a lesser extent, suggsting that it may achieve a good balance between parsability and simplicity.

As the whole trends of the two parsers are similar, we mainly foucs on RBG in the analysis below.

What kinds of errors are reduced by our conversion? To inspect this, we compare F1-scores of each arc label. Table 3 summarizes the results for the frequent labels, and interestingly we can see that the improvements are observed for more semantically crucial, core relations such as dobj (+0.81), nmod (+2.34), and nsubj (+2.01).[6] This is not surprising as these relations are involved in most of our conversion. See Figure 1, on which in the original tree, nmod arc connects two content words (*went* and *bar*) while in the converted tree, they are connected via a function word *to*. The result suggests that this latter structure is more parsable than the original one, possibly because directly connecting content words is harder due to the sparsity. We further investigate this hypothesis quantitatively later.

The F1-scores degrade in some functional lables, such as mark (-2.74) and case (-0.85). In-

[3] https://sourceforge.net/projects/mstparser/

[4] https://github.com/taolei87/RBGParser

[5] We exclude Arabic and French since they caused problems in training with RBG in a preliminary study.

[6] In the following, by *core labels* we mean the labels in the "core" row at Table 3 while by *functional labels* we mean the other labels (func).

L.	UAS				LAS				CNC	
	MST		RBG		MST		RBG		RBG	
	UD	CONV	UD	CONV	UD	CONV	UD	CONV	UD	CONV
bg	88.39	**88.86**	90.33	**90.74**	81.63	**82.63**	84.85	**85.64**	80.74	**81.92**
cs	86.65	**87.20**	91.40	**91.67**	79.85	**80.65**	87.25	87.22	85.23	85.21
da	82.03	**83.46**	86.08	**86.51**	76.81	**78.52**	82.13	**82.65**	78.42	**79.54**
de	84.69	84.66	**87.19**	86.68	75.47	**77.69**	79.39	**80.63**	72.03	**74.10**
en	85.97	**86.30**	89.69	89.65	80.67	**81.89**	86.32	**86.50**	82.30	**82.83**
es	85.98	**86.47**	89.02	**89.21**	80.13	**81.95**	84.98	**85.75**	77.33	**79.00**
et	**81.04**	80.81	87.67	87.60	71.28	**71.56**	83.84	**84.07**	82.58	**82.99**
fa	83.26	**84.25**	82.83	**84.37**	78.43	**80.10**	78.64	**80.56**	74.53	**77.47**
fi	**76.76**	76.42	85.57	**85.80**	68.24	**68.55**	81.69	**82.46**	80.46	**81.22**
hi	89.80	**92.14**	**95.10**	94.99	84.11	**87.20**	**91.74**	90.76	**87.96**	87.22
hu	79.31	**79.94**	**84.53**	84.15	66.47	**67.26**	79.53	**79.94**	77.19	**78.06**
it	88.82	**89.48**	92.14	**92.83**	83.90	**85.94**	89.22	**90.25**	83.31	**85.27**
ja	87.67	**90.20**	91.58	**92.24**	79.96	**85.41**	87.70	87.62	81.09	81.14
no	89.14	**89.44**	91.57	91.57	84.06	**85.23**	88.31	88.32	84.81	**85.14**
pl	**88.10**	87.71	92.25	**92.47**	80.20	**80.73**	87.51	**87.70**	85.08	**85.64**
pt	**85.82**	85.34	90.51	**91.04**	80.16	**80.53**	86.79	**87.47**	80.30	**81.90**
ru	81.46	**81.91**	86.76	**87.13**	74.79	**75.86**	83.15	**83.92**	81.01	**82.04**
tr	**79.02**	78.90	85.10	85.13	62.56	**62.66**	75.33	**75.57**	73.70	**74.19**
zh	**79.28**	79.07	**85.75**	85.48	73.44	**74.72**	80.91	**81.68**	79.43	**80.45**
Avg.	84.38	**84.87**	88.69	**88.91**	76.96	**78.37**	84.17	**84.67**	80.40	**81.33**

Table 2: Comparison of unlabelled (UAS) and labelled (LAS) attachment scores. See body for CNC. A bold score means that the difference is more than 0.1 points.

Type	Label	Ratio	UD	CONV
core	advmod	4.9%	79.24	79.15
	amod	6.3%	92.41	92.46
	conj	4.4%	66.56	**68.07**
	dobj	5.7%	81.92	**82.73**
	nmod	14.6%	76.52	**78.86**
	nsubj	7.3%	80.19	**82.20**
func	case	11.4%	**95.54**	94.69
	cc	3.3%	79.47	**80.00**
	det	6.6%	94.99	94.95
	mark	2.9%	**87.39**	84.65

Table 3: F1-scores (UD and CONV) and the average ratio in the test set (Ratio) of the frequent labels.

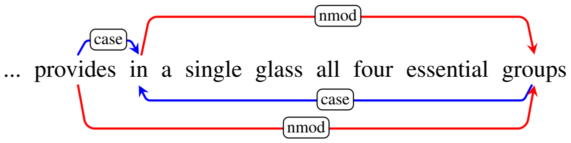

Figure 2: A failed output of CONV model (above), which induces an additional error on case with the backward conversion (below).

specting the outputs, we find that this essentially arises in our backward conversion, which induces errors on these arcs even when they are correctly attached in the (CONV) parser output, if another *core* label arc following them, such as nmod, attaches wrong. Figure 2 describes the situation. In the initial parser output (above), the case arc to *in* is correct although it misattaches *groups* as a child of *in* (the correct head is *provides*). By the backward conversion, then, it induces a wrong case arc from *groups* to *in*, which hurts both precision and recall. In summary, we can say that just predicting correct functional arcs (e.g., case) is equally easy for both representations, but our method needs correct analysis on *both* functional and core arcs, to recover the true functional arcs.

Although this additional complexity seems deficiency, the oveall scores (FAS) increase, which suggests that the majority case is successful predictions of both arcs thanks to our conversion. In other words, though our method slightly drops scores of functional arcs, it saves much more arcs of core relations, which are generally harder.

CNC To further verify the intuition above, now we introduce another metric called the CNC score, which is recently proposed in Nivre (2016) for UD evaluation purpose and calculates LAS excluding functional arcs[7]. The last column in Table 2 shows the results, where the improvements are clearer than LAS, +0.9 points on average. The results confirm the above observation that our method facilitates to find core grammatical arcs at a slight sacrifice of functional arcs.

Head word vocabulary entropy Finally, we provide an analysis to answer the question *why* our method improves the scores of core dependency arcs. As we mentioned above, this may be relevant to the ease of sparseness by placing function words between two content words. We verify this intuition quantitatively in terms of the entropy reduction of head word vocabulary. Schwartz et al. (2012) hypothesize about such correlation between entropy and parsability, although no qunatitative verification has been carried out yet.

For each dependency $h \overset{l}{\curvearrowright} w$ from h to w with label l in the training data, we extract a pair $((p, l, w), h)$ where p is the POS tag of w. We then discard the pairs such that a tuple (p, l, w) appears less than five times, and calculate the entropy of head word, $H_l(h)$ from the conditional probablity $P(h|p, l, w)$. We perform this both for the original UD and converted data, and calculate the difference for each label $H_l^{orig}(h) - H_l^{conv}(h)$.

See Figure 3 above, where many nmods appear on the upper left side, meaning that the reduction of entropy contributes to the larger improvements cross-linguistically. Other points on this area include dobjs of Japanese and Persian, both of which employ case constructions for expressing objects.

We also explore the correlation between LAS and the averaged reduction of entropy per a token in each language. Figure 3 below shows a negative correlation, which means the reduction of entropy as a whole by the conversion relates with the overall improvement. In particular in MST, we find a strong negative correlation ($r = -.75; p < .01$). RBG, on the other hand, has a weaker, non-significant negative correlation ($r = -.35; p = .14$) when excluding Hindi, which seems an outlier. These correlations imply that the variation of entropy can be a metric of assessing an annotation framework, or a conversion method.

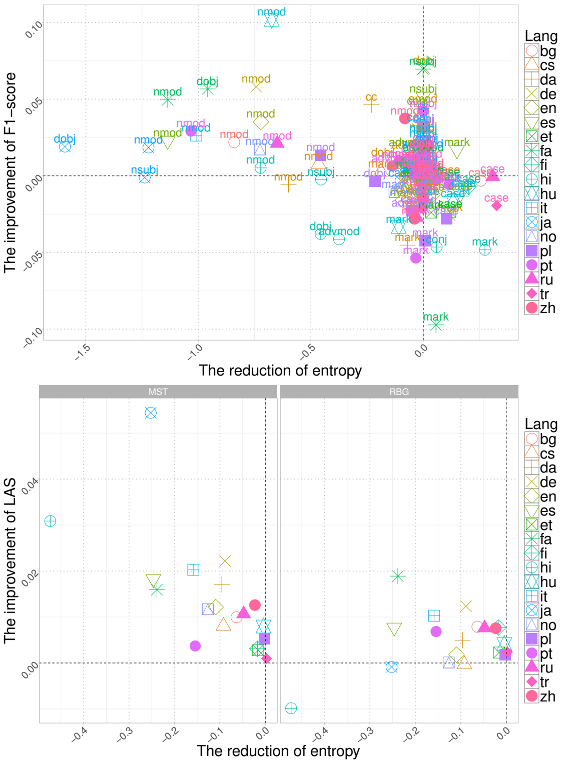

Figure 3: The reduction of entropy and the improvement of F1-score (above) and LAS (below)

4 Conclusion and Future Work

We have shown that our back-and-forth conversion around function words reduces head word vocabulary, leading to improvements of parsability and labelled attachment scores. This is the first empirical result on UD showing the parser preference to the function head scheme across languages. The method is modular, and can be combined with any parsing systems as pre- and post-processing steps.

Recently there has been a big success in the transition-based neural dependency parsers, which we have not tested mainly because the most such systems currently available, such as SyntaxNet (Andor et al., 2016) and LSTMParser (Dyer et al., 2015), do not support non-projective parsing. The neural parsers are advantageous in that the bilexical sparsity problem, the main challenge in UD parsing for the ordinary feature-based systems, might be alleviated thanks to word embeddings. It is thus an interesting and important future work to develop a neural dependency parser designed for non-projective parsing and see whether our conversion is still effective for such stronger system.

[7] Arcs with the following relations: aux, auxpass, case, cc, cop, det, mark, and neg.

Acknowledgements

This work was in part supported by JSPS KAKENHI Grant Number 16H06981.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August. Association for Computational Linguistics.

Marie-Catherine de Marneffe and Christopher D Manning. 2008. The stanford typed dependencies representation. In *COLING Workshop on Cross-framework and Cross-domain parser evaluation*, pages 1–8.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May. European Language Resources Association (ELRA). ACL Anthology Identifier: L14-1045.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

Jan Hajic, Barbora Vidová-Hladká, and Petr Pajas. 2001. The prague dependency treebank: Annotation structure and support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114.

Angelina Ivanova, Stephan Oepen, and Lilja Øvrelid. 2013. Survey on parsing three dependency representations for english. In *51st Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*, pages 31–37, Sofia, Bulgaria, August. Association for Computational Linguistics.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of the 16th Nordic Conference of Computational Linguistics*, pages 105–112.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1381–1391, Baltimore, Maryland, June. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August. Association for Computational Linguistics.

Jens Nilsson, Joakim Nivre, and Johan Hall. 2006. Graph transformations in data-driven dependency parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 257–264, Sydney, Australia, July. Association for Computational Linguistics.

Joakim Nivre. 2016. Universal depndency evaluation. In *http://stp.lingfil.uu.se/ nivre/docs/udeval-cl.pdf*.

Rudolf Rosa. 2015. Multi-source cross-lingual delexicalized parser transfer: Prague or stanford? In *Proceedings of the Third International Conference on Dependency Linguistics*, pages 281–290.

Roy Schwartz, Omri Abend, and Ari Rappoport. 2012. Learnability-based syntactic annotation design. In *Proceedings of COLING 2012*, pages 2405–2422, Mumbai, India, December. The COLING 2012 Organizing Committee.

Natalia Silveira and Christopher Manning. 2015. Does universal dependencies need a parsing representation? an investigation of english. In *Proceedings of the Third International Conference on Dependency Linguistics*, pages 310–319.

Daniel Zeman, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. 2012. Hamledt: To parse or not to parse? In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eighth International Conference on Language Resources* *and Evaluation (LREC-2012)*, pages 2735–2741, Istanbul, Turkey, May. European Language Resources Association (ELRA). ACL Anthology Identifier: L12-1223.