A Unifying Theory of Transition-based and Sequence Labeling Parsing

Carlos Gómez-Rodríguez **Michalina Strzyz** **David Vilares**
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Research Group
Departamento de Ciencias de la Computación y Tecnologías de la Información
Campus de Elviña, s/n, 15071 A Coruña, Spain
{carlos.gomez,michalina.strzyz,david.vilares}@udc.es

Abstract

We define a mapping from transition-based parsing algorithms that read sentences from left to right to sequence labeling encodings of syntactic trees. This not only establishes a theoretical relation between transition-based parsing and sequence-labeling parsing, but also provides a method to obtain new encodings for fast and simple sequence labeling parsing from the many existing transition-based parsers for different formalisms. Applying it to dependency parsing, we implement sequence labeling versions of four algorithms, showing that they are learnable and obtain comparable performance to existing encodings.

1 Introduction

Transition-based parsing algorithms compute the syntax or semantics of sentences as graphs; for instance in the form of dependency (Fraser, 1989; Yamada and Matsumoto, 2003; Nivre et al., 2004), phrase-structure (Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013) or meaning representations (Wang et al., 2015; Swayamdipta et al., 2016; Hershcovich et al., 2018).

These algorithms define an abstract state machine where each state (configuration) holds a structured representation, as well as auxiliary data structures (often, but not always, a buffer and a stack of tokens). Shift-reduce actions (transitions) are defined to move the system between states until a full parse is found. The transition to take at each state was traditionally predicted by data-driven classifiers based on local decisions and rich feature representations (Zhang and Nivre, 2011). With the adoption of deep learning, which can globally contextualize word representations, the dependency on hand-crafted features has been drastically reduced (Kiperwasser and Goldberg, 2016; Shi et al., 2017); and it has also been shown that alternative ways to attack the problem can be practical.

More particularly, several parsing problems have been cast as a machine translation task, where a sequence-to-sequence (seq2seq) network maps the sentence into a string of arbitrary length that encodes a linearized graph (Vinyals et al., 2015; Li et al., 2018; Konstas et al., 2017). To a certain extent, the attention mechanism in these seq2seq models can be seen as an abstraction of the stack and buffer in transition-based parsers, where the attention weights mark the relevant words to generate the next component of the output string. Alternatively, some authors have reduced constituent and dependency parsing to sequence labeling, where given an input sentence of length n, the output has length n too, assigning one label to each word (Gómez-Rodríguez and Vilares, 2018; Strzyz et al., 2019). However, these reductions have consisted in defining custom encodings for the output structure, which cannot be automatically derived.

In this context, some studies have linked transition-based parsers to seq2seq architectures, as in (Li et al., 2018), but to the best of our knowledge there is no unified framework or theory for transition-based and sequence labeling parsing.

Contribution (i) Our first contribution is theoretical, connecting the transition-based and sequence labeling parsing paradigms. We give a broad definition of a left-to-right transition system, covering the

Proceedings of the 28th International Conference on Computational Linguistics, pages 3776–3793
Barcelona, Spain (Online), December 8-13, 2020

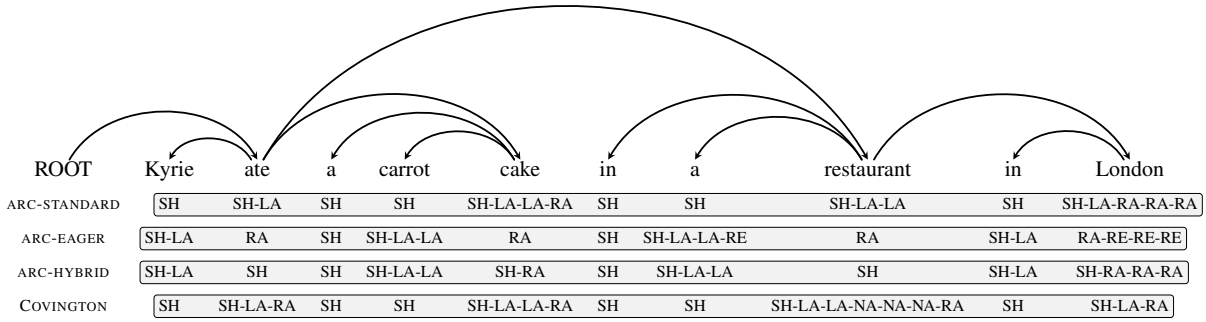

Figure 1: Examples of label distribution based on the sequence of actions for the corresponding transition systems. LA: Left Arc, RA: Right Arc, SH: Shift, NA: No-Arc. The last shift action in the sequence (which just terminates the computation and is always present) is omitted in the Covington system.

majority of transition-based parsers, and prove that the transitions produced by such systems for a sentence of length n can be mapped to a sequence of n labels, hence providing a mapping from transition-based parsers to sequence labeling parsers. (ii) The second contribution is empirical, applied to dependency parsing. We implement projective and non-projective transition-based algorithms, cast them in a sequence labeling setup and show that they are learnable, and even outperform some existing custom encodings for parsing as labeling. The source code is available at `https://github.com/mstrise/dep2label`.

2 Preliminaries

Let $\mathbf{w} = w_1 \dots w_n$ be an input sentence. We will denote by $\mathbb{P}_\mathbf{w}$ the set of possible well-formed parses for $\mathbf{w}$ in the relevant parsing formalism (e.g. in projective dependency parsing, $\mathbb{P}_\mathbf{w}$ is the set of projective dependency trees of n nodes).

Following Nivre (2008), with some adaptations to generalize the notions beyond dependency parsing, we can define a **transition system** as a quadruple $S = (C, T, c_s, C_t)$ where:

- C is a set of configurations, such that each configuration $c \in C$ contains at least a partially-built parse P_c (be it a partial syntactic dependency tree, semantic dependency tree, constituent tree, or any other structure that can be built using the input words), in addition to any other data structures needed by each specific transition system,
- T is a set of transitions, i.e. partial functions $t : C \to C$,
- c_s is an initialization function, mapping an input sentence $\mathbf{w} = w_1 \dots w_n$ to an initial configuration in C,
- $C_t \subseteq C$ is a set of terminal configurations.

A transition system parses an input sentence $\mathbf{w}$ by starting in the initial configuration $c_s(\mathbf{w})$, and applying a sequence of transitions until a final configuration $c_f \in C_t$ is reached. At that point, the parse $P_{c_f} \in \mathbb{P}_\mathbf{w}$ is returned.

Thus, for a transition system $S = (C, T, c_s, C_t)$, we can define a **computation** of S on $\mathbf{w}$ as a sequence of configurations $c_0, c_1, \dots, c_m$ such that each c_i is obtained from c_{i-1} by applying a transition t_i. Such a computation is **complete** for $\mathbf{w}$ if $c_0 = c_s(\mathbf{w})$ and $c_m \in C_t$. We denote by $\mathbb{C}^S_\mathbf{w}$ the set of complete computations of S on $\mathbf{w}$. Note that a computation is uniquely determined by its starting configuration and the sequence of transitions $t_1, \dots, t_m$ applied to it. A complete computation is uniquely determined by the sequence of transitions $t_1, \dots, t_m$, as the initial configuration is fixed.

Finally, we define a **static oracle** for a transition system $S = (C, T, c_s, C_t)$ as a function $\omega : \mathbb{P}_\mathbf{w} \to \mathbb{C}^S_\mathbf{w}$ such that for every $P \in \mathbb{P}_\mathbf{w}$, the parse associated with the final configuration in $\omega(P)$ is P. That is, given

a gold parse for the sentence $\mathbf{w}$, a static oracle returns a (canonical) computation of S that produces that gold parse on $\mathbf{w}$. Note that a correct transition system should be able to produce all the possible well-formed parses in $\mathbb{P}_{\mathbf{w}}$, and hence, a static oracle must exist.[1]

3 Mapping transition-based parsers to sequence labeling parsers

We first formally define our mapping from transition systems to sequence labeling parsers, to then explain it in detail, provide examples, and analyze how it applies to different transition systems:

Definition 1. Let $S = (C, T, c_s, C_t)$ be a transition system. We say that S is a **left-to-right transition system** if there is a subset of transitions $T_s \subseteq T$, called the **read transitions**, which satisfy the following conditions:

1. Every sequence of transitions $t_1, \ldots, t_m$ corresponding to a complete computation for a sentence $w_1 \ldots w_n$ has exactly n read transitions, one of which is t_1.

2. There is a constant value k such that, for each $1 \leq i \leq n$, the partial parse contained in a computation starting at c_s and containing i read transitions in its transition sequence is a partial parse over the substring $w_1 \ldots w_{i+k}$.

Such parsers can be mapped into a sequence labeling encoding as follows:

Definition 2. Let $S = (C, T, c_s, C_t)$ be a left-to-right transition system. Let $\gamma = c_0, \ldots, c_m$ be a complete computation of S on a sentence $w_1 \ldots w_n$. By the first condition of a left-to-right transition system, we know that the sequence of transitions of γ is of the form

$$t_1^r, t_1^1, \ldots, t_1^{m_1}, t_2^r, t_2^1, \ldots, t_2^{m_2}, \ldots, t_n^r, t_n^1, \ldots, t_n^{m_n}$$

where each t_i^r is a read transition, and t_i^j is the jth consecutive non-read transition after t_i^r.

Then, we define the **label sequence** associated to γ, denoted $L(\gamma)$, as the sequence of n labels where the ith label is $t_i^r, t_i^1, \ldots, t_i^{m_i}$.

Thus, informally speaking, the notion of a transition system being left-to-right corresponds to the presence of a transition (or set of transitions) that read a new word from the input, in left-to-right order. In transition systems where such transitions are present, processing a sentence of length n requires n of them (as each of the n words must be read from the input); and we can use this property to split transition sequences for full trees into n labels (one per word), each led by a read transition. The constant k in Definition 1 is an offset between the number of read transitions that have been executed at each given state and the words that can appear in parses at that state, as seen in the examples below.

Thus, the above definition provides a mapping from complete computations of a transition system to a label sequence. Since, as explained in Section 2, every correct transition system has at least one static oracle that maps gold parses to complete computations, it is easy to compose both mappings to define a mapping from left-to-right transition systems to sequence labeling parsers, where labels are subsequences of transitions:

Definition 3. Let $S = (C, T, c_s, C_t)$ be a left-to-right transition system, $\mathbf{w} = w_1 \ldots w_n$ a sentence of length n, and ω a static oracle for S. Then, we define the **sequence labeling encoding** associated to S as the mapping $\kappa : \mathbb{P}_{\mathbf{w}} \rightarrow (T^*)^n$ such that for each $P \in \mathbb{P}_{\mathbf{w}}$, $\kappa(P) = L(\omega(P))$.

3.1 Examples

The arc-standard transition system for projective dependency parsing (shown in Table 1) is a left-to-right transition system, where SH is the only read transition. Figure 1 shows how a parse for an example projective sentence is converted into n labels. In this transition system, $k = 0$ because it needs to place

[1] The traditional definition of a static oracle was as a function that returns one transition at a time, so that the computation is obtained by applying the oracle repeatedly until a final configuration is reached. However, as argued by Goldberg and Nivre (2012), such functions only apply to transitions in the canonical transition sequence defined by the oracle, so static oracles are only correct as functions from gold parses to computations as considered here.

Transition systems	
Initial config $c_s(w_1 \dots w_n)$: $([root], [1...n], \emptyset)$ Final configs: $C_t = \{([], [], P)\}$	Initial config $c_s(w_1 \dots w_n)$: $([], [], [root, 1...n], \emptyset)$ Final configs: $C_t = \{(\lambda_1, \lambda_2, [], P)\}$
ARC-STANDARD	COVINGTON NON-PROJECTIVE
SH $(\sigma, b\|\beta, P) \Rightarrow (\sigma\|b, \beta, P)$	SH$_c$ $(\lambda_1, \lambda_2, b\|\beta, P) \Rightarrow (\lambda_1 \cdot \lambda_2\|b, [], \beta, P)$
LA$_{st}$ $(\sigma\|s_1\|s_0, \beta, P) \Rightarrow (\sigma\|s_0, \beta, P \cup \{s_0 \rightarrow s_1\})$	NO-ARC$_c$ $(\lambda_1\|s, \lambda_2, \beta, P) \Rightarrow (\lambda_1, s\|\lambda_2, \beta, P)$
RA$_{st}$ $(\sigma\|s_1\|s_0, \beta, P) \Rightarrow (\sigma\|s_1, \beta, P \cup \{s_1 \rightarrow s_0\})$	LA$_c$ $(\lambda_1\|s, \lambda_2, b\|\beta, P) \Rightarrow (\lambda_1, s\|\lambda_2, b\|\beta, P \cup \{b \rightarrow s\})$ if $\nexists k : k \rightarrow s \in P$ (single head) if $\nexists$ path $s \rightarrow ... \rightarrow b$ in P (acyclicity)
ARC-EAGER	RA$_c$ $(\lambda_1\|s, \lambda_2, b\|\beta, P) \Rightarrow (\lambda_1, s\|\lambda_2, b\|\beta, P \cup \{s \rightarrow b\})$ if $\nexists k : k \rightarrow b \in P$ (single head) if $\nexists$ path $b \rightarrow ... \rightarrow s$ in P (acyclicity)
SH $(\sigma, b\|\beta, P) \Rightarrow (\sigma\|b, \beta, P)$	
LA$_e$ $(\sigma\|s, b\|\beta, P) \Rightarrow (\sigma, b\|\beta, P \cup \{b \rightarrow s\})$ if $\nexists k : k \rightarrow s \in P$	
RA$_e$ $(\sigma\|s, b\|\beta, P) \Rightarrow (\sigma\|s\|b, \beta, P \cup \{s \rightarrow b\})$	
REDUCE $(\sigma\|s, \beta, P) \Rightarrow (\sigma, \beta, P)$ if $\exists k : k \rightarrow s \in P$	
ARC-HYBRID	
SH $(\sigma, b\|\beta, P) \Rightarrow (\sigma\|b, \beta, P)$	
LA$_e$ $(\sigma\|s, b\|\beta, P) \Rightarrow (\sigma, b\|\beta, P \cup \{b \rightarrow s\})$	
RA$_{st}$ $(\sigma\|s_1\|s_0, \beta, P) \Rightarrow (\sigma\|s_1, \beta, P \cup \{s_1 \rightarrow s_0\})$	

Table 1: Transitions in arc-standard, arc-eager, arc-hybrid and Covington with their initial and final configurations.

words on the stack (via a read transition) before creating dependencies between them, so after having executed i read transitions, the words available for parsing are restricted to $w_1 \dots w_i$. For example, one needs two SH transitions to be able to link "Kyrie" as dependent of "ate".

The arc-eager transition system for projective dependency parsing (also in Table 1) is a left-to-right transition system, where both SH and RA are read transitions. This shows that, while in most transition systems the notion of a read transition is implemented by a SH or shift transition (which moves a word from the remaining input into a stack or another data structure for words being processed), this need not always be the case: for example, in arc-eager, the right-arc transition also moves a word from the remaining input into the stack apart from creating a right arc. Figure 1 shows the n labels resulting from the parse for the example sentence. In this transition system, $k = 1$ because it can create dependencies involving the first word in the buffer, so computations with i read transitions can result into a partial parse involving $w_1 \dots w_{i+1}$. For example, the link between "Kyrie" and "ate" in Figure 1 is created after only one read transition.

3.2 Discussion

Informally speaking, the first condition of a left-to-right transition system simply states that when parsing a sentence of length n, the parser should start with a read transition and execute exactly n read transitions in total. This does not impose any limits on the nature of the transitions but only on their number and arrangement in transition sequences, and it is the basis of our mapping, which uses the n read transitions to split the transition sequence into n subsequences that can act as labels.

The second condition means that the algorithm needs to proceed in an incremental left-to-right manner, driven by the read transitions, in the sense that each read transition introduces the possibility of using a new word in the parse, and words are introduced in left-to-right order. This generalizes the classic notion of shift transitions reading words and placing them into a data structure (like a stack) that then allows their manipulation. Note that this is a weak form of incrementality. For example, in an arc-standard dependency parser a chain of right arcs is processed from right to left, but this condition is still met. The constant k is typically 0 or 1 in most parsers in the literature. For example, it is 1 in the classic arc-eager dependency parser and 0 in arc-standard, as explained above.

Moreover, we wish to stress that the notion of a left-to right transition system defined in this work only implies incrementality at the transition system level. This means that it does not guarantee that any implementation of a such transition system is left-to-right incremental in the traditional psycholinguistic sense, as the definition of a transition system only concerns the transitions available and cannot impose restrictions on what a particular implementation can use to decide between them. For instance, imple-

mentations that use BiLSTMs (as those in our own experiments below) are not left-to-right incremental, as they have information about future input encoded in the hidden representations of the input tokens. However, in that case, the absence of left-to-right incrementality is due to the word representations chosen for the implementation (and not due to the transition system); and any transition system that meets our definition can be implemented in a truly incremental way by using word representations that do not depend on future input.

Strictly speaking, the first condition in our definition of a left-to-right transition system is enough to define a mapping from transition sequences to sequences of n labels, thus obtaining a sequence labeling encoding. The definition of the mapping above does not formally use or rely on the second condition. However, we include said condition because it means that the label for each word will encode information about the transitions executed after reading that word. In other words, it means that not only the transition sequences of the parser can be encoded as sequences of n labels, but also that the ith label in the encoding is semantically linked to the ith word. We believe that this is a reasonable common-sense assumption for the resulting sequence labeling model to be learnable.

An example of a transition system that fails the second condition is the easy-first parser of Goldberg and Elhadad (2010). The algorithm runs n transitions in total that attach each input word to a head, so if we ignore the second condition, all its transitions could be considered read transitions, and we would obtain a compact sequence labeling encoding where each label contains a single transition. However, the problem is that the label for a given word is not semantically linked to that word, as the parser can create arcs in any order, so the encoding can hardly be considered practical.

3.3 Coverage

Algorithm	L2R?	Read t.	k	Algorithm	L2R?	Read t.	k
Arc-standard (Fraser, 1989; Nivre, 2004)	Yes	SH	0	Spinal arc-eager (Ballesteros and Carreras, 2015)	Yes	SH, RA	1
Arc-eager (Nivre, 2003)	Yes	SH, RA	1	Yamada and Matsumoto (2003)	No		
Arc-hybrid (Kuhlmann et al., 2011)	Yes	SH	1	Choi and Palmer (2011)	Yes	SH	1
Covington projective (Covington, 2001; Nivre, 2008)	Yes	SH	0	Choi and McCallum (2013)	Yes	No-SH, R-SH	1
Covington non-projective (Covington, 2001; Nivre, 2008)	Yes	SH	0	Non-monotonic arc-eager (Honnibal et al., 2013)	Yes	SH, RA	1
Easy-first (Goldberg and Elhadad, 2010)	No			Improved non-monotonic arc-eager (Honnibal and Johnson, 2015)	No		
Attardi (Attardi, 2006)	Yes	SH	0	Non-monotonic Covington (Fernández-González and Gómez-Rodríguez, 2017)	Yes	SH	1
Planar (Gómez-Rodríguez and Nivre, 2010)	Yes	SH	1	Tree-constrained arc-eager (Nivre and Fernández-González, 2014)	No		
2-Planar (Gómez-Rodríguez and Nivre, 2010)	Yes	SH	1	Non-local Covington (Fernández-González and Gómez-Rodríguez, 2018)	Yes	SH	1
Arc-eager with buffer transitions (Fernández-González and Gómez-Rodríguez, 2012)	Yes	SH	2,3	Two-register (Pitler and McDonald, 2015)	No		
Swap (Nivre, 2009)	No			Stack-pointer (Ma et al., 2018)	No		
Swap-hybrid (de Lhoneux et al., 2017b)	No			Left-to-right pointer network (Fernández-González and Gómez-Rodríguez, 2019b)	No		
Arc-swift (Qi and Manning, 2017)	Yes	SH, RA_k	1				

Table 2: Transition-based dependency parsers, whether they are left-to-right (L2R?) or not, read transitions in case they are, and value of the constant k. The value of k should be seen as a guide only, as k can vary between variants of each parser. For example most definitions of arc-standard create arcs between nodes on the stack, so $k = 0$ (Nivre, 2004) but it has also been defined in an equivalent form where arcs are created between stack and buffer, so $k = 1$ (see (Nivre, 2008)). The same happens with Attardi and other algorithms.

Table 2 shows different transition systems from the dependency parsing literature and whether they are

Algorithm	L2R?	R. t.	k	Algorithm	L2R?	R. t.	k
Bottom-up (Sagae and Lavie, 2005)	Yes	SH	0	In-order (Liu and Zhang, 2017b)	Yes	SH	0
Bottom-up with terminate action (Zhang and Clark, 2009)	Yes	SH	0	Discontinuous easy-first (Versley, 2014)	No		
Bottom-up with padding (Zhu et al., 2013)	Yes	SH	0	Swap (Maier, 2015)	No		
Odd-even (Mi and Huang, 2015)	Yes	SH	0	Skip-shift (Maier and Lichte, 2016)	No		
LR-inspired (Crabbé, 2014)	Yes	SH	0	Stanojević and G. Alhama (2017)	No		
Span-based (Cross and Huang, 2016)	Yes	SH	0	SR-GAP (Coavoux and Crabbé, 2017)	Yes	SH	0
Coavoux and Crabbé (2016)	Yes	SH	0	ML-GAP and ML-GAP-LEX (Coavoux et al., 2019)	Yes	SH	0
Non-binary bottom-up (Fernández-González and Gómez-Rodríguez, 2019a)	Yes	SH	0	Coavoux and Cohen (2019) stack-free system	Yes	SH	0
Top-down (Dyer et al., 2016)	Yes	SH	0	Tetra-tagging (Kitaev and Klein, 2019)	Yes	$\nwarrow, \nearrow$	0

Table 3: Transition-based constituent parsers, whether they are left-to-right (L2R?) or not, read transitions in case they are, and value of the constant k.

left-to-right or not, together with the values of the read transitions and k if applicable. As can be seen in the table, the majority of known transition-based parsers conform to our definition of left-to-right, and hence yield an encoding that can be used to define a sequence labeling parser with the framework defined here. Exceptions are parsers with multiple left-to-right passes over the input (Yamada and Matsumoto, 2003), those that can create arcs between words in arbitrary order (like the aforementioned easy-first parser of Goldberg and Elhadad (2010)) those that use unshift transitions that return a node to the buffer (Nivre and Fernández-González, 2014; Honnibal and Johnson, 2015) or swap transitions which also have that side effect (Nivre, 2009), and those that can create arcs involving nodes arbitrarily far to the right (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019b).

For all of these exceptions, one could still construct sequence-labeling encodings by ignoring the second condition of a left-to-right system as arc-creating transitions always meet the first condition, but this is dubiously practical in most cases, as explained above. A singular case in this respect is the left-to-right pointer network parser of Fernández-González and Gómez-Rodríguez (2019b). This parser does not fit the second condition of our definition because it can use nodes arbitrarily to the right as heads (in spite of being purely left-to-right in terms of the order in which it considers dependents); and if we still apply our transformation to it, we obtain an encoding isomorphic to the relative positional encoding used in Li et al. (2018), which has been shown useful under strong machine learning models (Vacareanu et al., 2020).

Table 3 shows transition-based constituency parsers with analogous information to Table 2. In this case, all of the listed parsers that do not fall into our definition of left-to-right are discontinuous constituent parsers that use swap transitions (Versley, 2014; Maier, 2015). Every continuous constituent parser that we found is covered, as well as discontinuous parsers that use other devices, like gap transitions (Coavoux et al., 2019) or set handling (Coavoux and Cohen, 2019). All of the supported constituent parsers have $k = 0$, following the traditional shift-reduce paradigm that only operates on nodes after reading them from the input buffer.

Algorithm	L2R?	Read t.	k	Algorithm	L2R?	Read t.	k
Sagae and Tsujii (2008)	Yes	SH	1	Online reordering (Zhang et al., 2016)	No		
Titov et al. (2009)	Yes	SH	1	Two-stack (Zhang et al., 2016)	Yes	SH	1
Ribeyre et al. (2014)	No			Hershcovich et al. (2017)	No		
Tokgöz and Eryiğit (2015)	Yes	SH	1	Wang et al. (2018)	Yes	No-SH, R-SH	1
Artsymenia et al. (2016)	Yes	SH	1	DM, PSD (Che et al., 2019)	Yes	SH	1

Table 4: Transition-based semantic dependency parsers, whether they are left-to-right (L2R?) or not, read transitions in case they are, and value of the constant k. For simplicity we only include semantic dependency parsers and exclude parsers for formalisms that go beyond dependency graphs, like AMR, and cross-framework parsers (e.g. Bai and Zhao (2019)).

Table 4 lists transition-based semantic dependency parsers with analogous information to Tables 2 and 3. Once again, parsers that do not fall into our definition of left-to-right are mostly those that define a swap transition. Note that for this coverage analysis we exclude semantic formalisms that go beyond

dependency graphs, such as AMR (Banarescu et al., 2013), where pure transition-based models (e.g. Damonte et al. (2017), Ballesteros and Al-Onaizan (2017),Vilares and Gómez-Rodríguez (2018)) need to remove tokens and also create (multiple) concepts from words, and therefore include specific transitions to do so. Removing tokens can be seen as a read transition, but creating many concept nodes from a single word breaks the left-to-right condition and does not ensure that the system will have n read transitions, where n is the length of the input sentence. Although outside the scope of this paper, note that a hybrid system that first computed the concepts from words (e.g. with a seq2seq architecture), to then apply a transition-based graph parser could support the left-to-right condition in the same way as other formalisms we have considered.

Putting it all together, our mapping is applicable to a wide range of transition systems, spanning a variety of formalisms: projective and non-projective dependency parsing, continuous and discontinuous constituency parsing, and various flavours of semantic parsing. Also, note that the impossibility to map buffer-based[2] swap transition-based parsers is not related to inability of our approach to handle non-projectivity, but to the non-left-to-right nature of those swap models. For instance, in Table 2 we showed how other non-projective transition-based algorithms (Covington, 2001; Attardi, 2006; Gómez-Rodríguez and Nivre, 2010) can be mapped to a sequence labeling encoding.

4 Experiments

To test the practical applicability of our theoretical contribution, we implement sequence labeling versions, obtained using the mapping in Section 3, of various syntactic dependency parsers. We include three well-known projective parsers: arc-standard (Nivre, 2004), arc-eager (Nivre, 2003) and arc-hybrid (Kuhlmann et al., 2011); and one parser with full coverage of non-projective trees: the Covington non-projective parser (Covington, 2001; Nivre, 2008). The transition systems for all of these parsers are shown in Table 1. For comparison, we also include two of the best encodings to the date for dependency parsing as labeling (Strzyz et al., 2019): (i) the rel-PoS (Spoustová and Spousta, 2010) and (ii) the bracketing encoding (Yli-Jyrä and Gómez-Rodríguez, 2017). The former casts the problem as a head-selection task where each token encodes its head using a PoS-tag-based offset. The latter assigns a label to each token encoding a set of incoming/outgoing arcs from that and neighbouring tokens.

4.1 Data

Following Anderson and Gómez-Rodríguez (2020), we choose a subset of UDv2.4 treebanks (Nivre and others, 2019) which includes languages with a variety of corpus sizes, language typologies, alphabets and levels of non-projectivity, among other differences. More particularly, these treebanks are: Ancient $\text{Greek}_{\text{Perseus}}$, $\text{Chinese}_{\text{GSD}}$, $\text{English}_{\text{EWT}}$, $\text{Finnish}_{\text{TDT}}$, $\text{Hebrew}_{\text{HTB}}$, $\text{Russian}_{\text{GSD}}$, $\text{Tamil}_{\text{TTB}}$, $\text{Uyghur}_{\text{UDT}}$ and $\text{Wolof}_{\text{WTB}}$. Appendix A shows the number of labels that our approach generates for each treebank. We also use UDpipe (Straka, 2018) to obtain data with predicted segmentation, tokenization and PoS tags.

4.2 Sequence labeling models

For training, we consider two sequence labeling encoders (see Appendix B for hyper-parameters), which will produce n hidden contextualized representations $\mathbf{h}_i$ to generate the labels:

BiLSTMs (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) We use the NCRF++ framework (Yang and Zhang, 2018) to train the models and consider two different setups:

1. A setup where the input vectors to the models are a concatenation of the pre-trained word embeddings by Ginter et al. (2017), a second word vector generated by a char-LSTM layer, which is trained together with the rest of the network, and PoS tag vectors.

2. Same as 1, but without PoS tag vectors.

[2]With buffer-based we refer to a swap transition that puts back a word from the stack to the buffer, contrary to stack-based swap that changes the order of the words in the stack, respecting the left-to-right condition.

Split	Encoding	Ancient Greek		Chinese		English		Finnish		Hebrew		Russian		Tamil		Uyghur		Wolof		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev_{s1}	rel-PoS	70.2	61.8	61.6	57.4	82.6	79.0	83.2	79.3	67.4	63.8	84.0	79.6	59.5	52.0	72.6	59.0	77.0	70.5	*73.1*	*66.9*
	Brackets	65.2	57.2	61.8	57.9	82.2	78.6	82.8	79.1	67.1	63.4	83.5	78.9	57.4	50.2	73.1	59.9	76.3	69.7	*72.2*	*66.1*
	Standard	62.2	54.7	60.3	56.2	79.9	76.3	79.7	76.0	64.2	60.5	80.0	75.8	57.8	49.7	72.5	59.2	73.0	66.8	*69.9*	*63.9*
	Eager	60.4	52.9	60.9	56.9	80.8	77.3	80.2	76.4	65.4	61.8	81.5	77.4	57.9	50.2	73.2	59.9	74.6	68.0	*70.5*	*64.5*
	Hybrid	62.7	55.2	60.3	56.3	80.5	76.9	80.4	76.7	65.2	61.6	81.1	76.4	58.1	50.5	72.9	59.5	72.9	66.5	*70.5*	*64.4*
	Covington	64.0	56.0	56.1	52.3	79.6	75.9	80.7	76.6	64.7	61.1	80.0	75.7	53.2	45.7	67.2	55.0	71.7	65.3	*68.6*	*62.6*
$test_{s1}$	rel-PoS	69.5	60.0	64.4	60.0	82.0	78.5	80.2	74.9	63.0	59.5	82.7	77.3	58.0	49.8	71.5	57.8	76.1	69.6	*71.9*	*65.3*
	Brackets	64.9	56.2	64.2	59.8	81.7	78.3	81.1	76.0	62.1	58.7	82.9	77.9	57.2	49.0	72.0	58.5	75.4	68.8	*71.3*	*64.8*
	Standard	61.0	52.6	62.9	58.5	79.2	75.8	78.5	73.4	60.3	56.5	79.1	74.2	57.1	48.6	71.5	58.2	72.0	65.7	*69.1*	*62.6*
	Eager	60.4	52.0	63.2	59.0	80.2	76.7	79.0	73.6	61.2	57.6	80.4	75.3	56.2	48.3	71.9	58.7	73.5	67.1	*69.6*	*63.1*
	Hybrid	62.1	53.5	63.2	58.2	79.5	76.1	78.9	74.0	60.4	56.8	79.3	74.4	57.0	48.7	71.3	58.6	72.2	65.7	*69.3*	*62.9*
	Covington	63.7	54.4	58.3	54.2	78.5	75.0	78.9	73.7	60.2	56.6	79.2	74.0	52.3	44.8	66.4	53.5	69.9	63.8	*67.5*	*61.1*
dev_{s2}	rel-PoS	65.3	58.3	58.8	55.3	80.3	77.1	80.8	77.3	65.3	62.2	81.3	77.4	52.7	41.9	65.7	53.3	73.1	66.3	*69.2*	*63.2*
	Brackets	64.7	57.2	63.8	59.4	83.4	80.0	84.1	80.4	68.7	65.0	84.0	79.7	55.9	45.1	72.1	58.8	75.2	67.4	*72.4*	*65.9*
	Standard	61.6	54.8	61.9	57.6	80.9	77.5	80.7	77.2	65.9	62.6	80.3	76.2	55.0	43.3	71.6	58.2	70.4	63.2	*69.8*	*63.4*
	Eager	60.0	53.2	63.4	59.2	81.9	78.5	81.6	78.2	66.9	63.2	81.7	77.4	56.4	45.5	72.3	58.8	73.4	66.1	*70.9*	*64.4*
	Hybrid	62.3	55.5	62.4	58.1	81.4	78.0	81.3	77.9	66.1	62.6	81.0	76.9	56.3	44.9	71.7	58.1	71.9	64.8	*70.5*	*64.1*
	Covington	63.7	56.1	57.4	53.5	80.5	76.8	81.8	78.0	66.3	63.0	80.0	75.7	51.1	40.8	67.0	54.1	71.3	63.6	*68.8*	*62.4*
$test_{s2}$	rel-PoS	62.9	55.1	60.3	56.8	78.5	75.3	65.6	59.6	59.7	56.6	79.0	73.8	51.6	40.6	64.9	52.4	72.3	65.6	*66.1*	*59.5*
	Brackets	63.4	54.8	65.3	61.1	82.4	78.9	75.1	67.7	62.3	58.6	81.8	75.8	56.9	42.6	71.0	57.4	75.1	67.2	*70.4*	*62.7*
	Standard	59.5	52.0	64.2	59.9	79.3	75.8	73.4	66.4	60.1	56.4	78.6	73.1	53.8	41.3	70.5	56.9	71.1	63.6	*67.8*	*60.6*
	Eager	59.5	51.7	64.5	60.5	80.0	76.6	72.3	64.1	61.1	57.3	79.7	74.0	53.4	41.1	71.6	57.9	72.9	65.2	*68.3*	*60.9*
	Hybrid	60.4	52.5	64.2	59.9	80.1	76.7	74.7	67.5	60.6	57.0	78.6	73.1	53.7	41.7	71.1	57.6	71.5	63.7	*68.3*	*61.1*
	Covington	62.5	53.5	59.6	55.7	78.6	74.9	73.8	66.6	60.6	57.0	78.2	72.5	48.2	36.6	66.3	53.6	69.2	61.6	*66.3*	*59.1*

Table 5: Results for the BiLSTM setups 1 (s1, with PoS tags) and 2 (s2, without PoS tags).

The motivation for setup 1 is that the PoS-tag-based encoding (Strzyz et al., 2019) requires PoS tags to decode the labels into a tree. Therefore, as PoS tags need to be computed to decode the tree, it is a fair argument to use them as input parameters, as done by Strzyz et al. (2019). The motivation for setup 2 is that in an era where the usefulness of PoS tags has been questioned (de Lhoneux et al., 2017a; Anderson and Gómez-Rodríguez, 2020), our encodings (which do not require PoS tags to decode the trees) could benefit in terms of speed and simplicity, with a minimum cost to accuracy.

BERT (Devlin et al., 2019) By default we fine-tune multi-lingual BERT (M-BERT) and in particular bert-base-multilingual-cased, except for English, Chinese and Finnish (Virtanen et al., 2019); for which monolingual models are available.[3,4] BERT splits the input into sub-word pieces (Wu et al., 2016) generating more sub-words than tokens, while we have a fixed amount of labels (equal to the number of tokens in the sentence) to assign. To solve this, we consider a word to be its first sub-word. Contrary to BiLSTMs, BERT does not use PoS tags as input, and we will only use them to decode the trees of the rel-PoS encoding.

To generate the output labels from the hidden vectors, we map each $\mathbf{h_i}$ to two output softmax layers (tasks) following a hard-sharing multi-task learning architecture: one that predicts the subsequence of transitions associated to that word, and a second one that predicts the dependency relation between that word and its head. The loss is computed as the sum of the categorical cross-entropy of both tasks. Corrupted sequences of predicted labels are postprocessed according to Appendix C.

4.3 Results

Table 5 shows the results of the encodings trained with the BiLSTM architecture. Overall, we see that the transition-based encodings perform comparable to preexisting encodings. The performance of arc-standard, arc-eager and arc-hybrid is similar across the board, while Covington suffers more, probably due to the large output vocabulary (see Appendix A) as a Covington transition sequence has $O(n^2)$ transitions

[3] `https://huggingface.co/transformers/pretrained_models.html`.

[4] bert-base-cased, bert-base-chinese, and TurkuNLP/bert-base-finnish-cased-v1

Split	Encoding	Chinese		English		Finnish		Hebrew		Russian		Tamil		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev	rel-PoS	59.8	57.0	82.3	79.9	82.6	79.7	61.8	58.0	77.6	73.2	43.4	33.6	*67.9*	*63.5*
	Brackets	69.3	65.3	87.2	84.4	88.6	85.1	66.6	61.7	80.6	75.4	49.8	36.8	*73.7*	*68.1*
	Standard	67.9	64.1	85.4	82.6	85.7	82.5	64.4	59.8	77.8	72.7	48.2	35.1	*71.6*	*66.1*
	Eager	67.0	63.2	85.3	82.4	86.8	83.5	64.7	59.6	77.8	72.9	49.7	35.2	*71.9*	*66.1*
	Hybrid	67.0	63.3	85.5	82.8	86.6	83.1	65.2	60.1	78.1	73.4	47.3	34.0	*71.6*	*66.1*
	Covington	59.7	56.0	81.4	78.8	82.4	79.2	61.9	57.3	73.9	68.9	44.6	31.6	*67.3*	*62.0*
test	rel-PoS	61.5	58.8	81.0	78.8	82.6	80.0	56.9	53.1	77.9	73.2	45.2	34.1	*67.5*	*63.0*
	Brackets	70.2	66.5	86.2	83.3	89.4	86.1	62.7	57.9	79.9	74.2	48.9	35.9	*72.9*	*67.3*
	Standard	68.5	64.7	84.0	81.2	86.1	83.1	59.6	54.3	77.6	72.3	51	37.3	*71.1*	*65.5*
	Eager	69.2	65.6	83.8	81.2	87.2	83.9	60.2	55.3	77.6	72.4	48.6	36.7	*71.1*	*65.8*
	Hybrid	69.1	65.5	84.1	81.3	86.6	83.6	60.3	55.7	77.5	72.3	48.2	35.6	*71.0*	*65.7*
	Covington	60.1	56.7	79.7	77.4	83.2	80.2	57.6	52.6	73.1	68.0	46.3	34.0	*66.7*	*61.5*

Table 6: Results for the BERT setup.

that we are grouping into n labels, contrary to the other three algorithms which run $O(n)$ transitions per sentence. Still, for highly non-projective treebanks such as Ancient-Greek-Perseus, the Covington mapping performs the best among the transition-based encodings; showing that it is learnable and yet preferable in this case to using pure projective transition-based encodings.

With respect to the BiLSTM setups 1 (with PoS tags) and 2 (without PoS tags), we observe that while the transition-based encodings (and the bracketing-based one) suffer little from the absence of PoS-tags, the rel-PoS based one greatly needs them, losing over 5 LAS points on average when they are not used. This is relevant since the use of PoS-tags increases the latency, and their usefulness has been questioned in some parsing setups (de Lhoneux et al., 2017a; Smith et al., 2018). In addition, in Appendix D we compute empirical upper bounds for the accuracy of the encodings by running them on the gold datasets (gold segmentation, tokenization and PoS tags) and compare them with the accuracy of the predicted setup with PoS tags. We can observe that the accuracy of upstream tasks has a large impact on all models, including the baselines, with the largest gaps between the parsing results on the predicted and gold dataset being observed for treebanks where segmentation, tokenization or tagging is difficult (for instance, the biggest difference is observed in Hebrew, where words and UPoS have the lowest prediction accuracy).

Table 6 shows the results for BERT. We excluded Ancient-Greek, Uyghur and Wolof since M-BERT does not support them, and we are not aware of a monolingual model. The tendency is similar to the BiLSTM setup 2, but with higher scores for monolingual BERTs (not M-BERT though) and rel-PoS lagging further behind the bracketing and the projective transition-based encodings (on average around 4 and 2.5 LAS points, respectively).

It is also worth comparing the results of our sequence labeling implementations of the projective transition-based parsers to the experiments of Shi et al. (2017) with regular transition-based implementations and different sets of positional features. For this, we run our BiLSTM setup on the same dataset as them, i.e., the English PTB dev set. The results can be seen in Table 7. Shi et al. (2017) concluded that, with a BiLSTM-based architecture, two positional features (one stack and one buffer feature) were needed to obtain reasonable accuracy in the arc-eager and arc-hybrid parsers, and three (two stack and one buffer feature) in the case of the arc-standard parser. In contrast, and although the exact accuracy numbers do not provide a homogeneous comparison due to hyperparameter differences, it can be seen in Table 7 that our labels of sequences of transitions can be learned with only one positional feature (our setup just assigns a sequence of labels beginning with a SH to each word shifted from the input, i.e., it only uses the first buffer word b_0, and has access to no explicit representation of stack elements at all). Thus, this shows that our multi-transition labels seem to be learnable with less data than individual transitions in transition systems, although the latter (with suitable features) are still ahead in terms of raw accuracy. While similar effects had been observed in seq2seq transition-based parsers (Zhang et al., 2017; Liu and Zhang, 2017a), these use more complex neural architectures than transition-based implementations, including attention weights that can focus on words in prominent stack positions (Liu and Zhang, 2017a). Here, we are just using plain BiLSTMs as in the standard transition-based implementation of Shi et al. (2017).

Model	Features	Arc-standard	Arc-eager	Arc-hybrid
Shi et al.	$\{s_2, s_1, s_0, b_0\}$	$93.95_{\pm 0.12}$	$93.92_{\pm 0.04}$	$94.08_{\pm 0.13}$
	$\{s_1, s_0, b_0\}$	$94.13_{\pm 0.06}$	$93.91_{\pm 0.07}$	$94.08_{\pm 0.05}$
	$\{s_0, b_0\}$	$54.47_{\pm 0.36}$	$93.92_{\pm 0.07}$	$94.03_{\pm 0.12}$
	$\{b_0\}$	$47.11_{\pm 0.44}$	$79.15_{\pm 0.06}$	$52.39_{\pm 0.23}$
Our model	$\{b_0\}$	92.13	93.22	92.66

Table 7: Performance (in UAS%) of our system using a single positional feature b_0 compared with Shi et al. (2017) on the English PTB dev set.

5 Conclusion

This paper has established a theoretical relationship between transition-based and sequence labeling parsing valid for a broad definition of left-to-right transition-based algorithms. It also provides a new set of encodings for sequence labeling parsing which are automatically derived, in contrast to existing ones which were created ad-hoc for this purpose. To test the practical utility, we ran experiments on dependency parsing for a diverse set of languages. Interestingly, the mapping is meaningful and learnable, despite not using any representation of stack nodes. While our experiments focused on dependency parsing as we only aimed to validate the theory, an obvious avenue for future work is to implement and test the sequence-labeling encodings derived from the constituent and semantic parsers in Tables 3 and 4. The latter are particularly relevant in the sense that, to our knowledge, there is no previous work on semantic parsing as sequence labeling; so our method provides the first encodings for this purpose.

Acknowledgments

This work has received funding from the European Research Council (ERC), which has funded this research under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from MINECO (ANSWER-ASAP, TIN2017-85160-C2-1-R), from Xunta de Galicia (ED431C 2020/11), and from Centro de Investigación de Galicia 'CITIC', funded by Xunta de Galicia and the European Union (European Regional Development Fund- Galicia 2014-2020 Program), by grant ED431G 2019/01. DV is supported by a 2020 Leonardo Grant for Researchers and Cultural Creators from the BBVA Foundation.

References

Mark Anderson and Carlos Gómez-Rodríguez. 2020. Distilling neural networks for greener and faster dependency parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 2–13, Online, July. Association for Computational Linguistics.

Mark Anderson and Carlos Gómez-Rodríguez. 2020. On the Frailty of Universal POS Tags for Neural UD Parsers. In *Proceedings of the 24th Conference on Computational Natural Language Learning*. To appear.

Artsiom Artsymenia, Palina Dounar, and Maria Yermakovich. 2016. IHS-RD-belarus at SemEval-2016 task 9: Transition-based Chinese semantic dependency parsing with online reordering and bootstrapping. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1207–1211, San Diego, California, June. Association for Computational Linguistics.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, New York City, June. Association for Computational Linguistics.

Hongxiao Bai and Hai Zhao. 2019. SJTU at MRP 2019: A transition-based multi-task parser for cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 86–94, Hong Kong, November. Association for Computational Linguistics.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark, September. Association for Computational Linguistics.

Miguel Ballesteros and Xavier Carreras. 2015. Transition-based spinal parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 289–299, Beijing, China, July. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August. Association for Computational Linguistics.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. HIT-SCIR at MRP 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76–85, Hong Kong, November. Association for Computational Linguistics.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria, August. Association for Computational Linguistics.

Jinho D. Choi and Martha Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 687–692, Portland, Oregon, USA, June. Association for Computational Linguistics.

Maximin Coavoux and Shay B. Cohen. 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 204–217, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 2016. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 172–182, Berlin, Germany, August. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 2017. Incremental discontinuous phrase structure parsing with the GAP transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain, April. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B. Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *Transactions of the Association for Computational Linguistics*, 7:73–89, March.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, New York, NY, USA. ACM.

Benoit Crabbé. 2014. An LR-inspired generalized lexicalized phrase structure parser. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 541–552, Dublin, Ireland, August. Dublin City University and Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas, November. Association for Computational Linguistics.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain, April. Association for Computational Linguistics.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017a. From raw text to universal dependencies - look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217, Vancouver, Canada, August. Association for Computational Linguistics.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017b. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104, Pisa, Italy, September. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, June. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2012. Improving transition-based dependency parsing with buffer transitions. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 308–319, Jeju Island, Korea, July. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2017. A full non-monotonic transition system for unrestricted non-projective parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 288–298, Vancouver, Canada, July. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. Non-projective dependency parsing with non-local transitions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 693–700, New Orleans, Louisiana, June. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019a. Faster shift-reduce constituent parsing with a non-binary, bottom-up strategy. *Artificial Intelligence*, 275:559 – 574.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019b. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 710–716, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Norman Fraser. 1989. Parsing and dependency grammar. *Ucl Working Papers in Linguistics 1: University College London*, pages 296–319.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December. The COLING 2012 Organizing Committee.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden, July. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium, October-November. Association for Computational Linguistics.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138, Vancouver, Canada, July. Association for Computational Linguistics.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 373–385, Melbourne, Australia, July. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September. Association for Computational Linguistics.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172, Sofia, Bulgaria, August. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Nikita Kitaev and Dan Klein. 2019. Tetra-tagging: Word-synchronous parsing with linear-time inference. *CoRR*, abs/1904.09745.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada, July. Association for Computational Linguistics.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA, June. Association for Computational Linguistics.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017a. Encoder-decoder shift-reduce syntactic parsing. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 105–114, Pisa, Italy, September. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017b. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia, July. Association for Computational Linguistics.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San Diego, California, June. Association for Computational Linguistics.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. 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 1202–1212, Beijing, China, July. Association for Computational Linguistics.

Haitao Mi and Liang Huang. 2015. Shift-reduce constituency parsing with dynamic programming and POS tag lattice. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1035, Denver, Colorado, May–June. Association for Computational Linguistics.

Joakim Nivre and Daniel Fernández-González. 2014. Squibs: Arc-eager parsing with the tree constraint. *Computational Linguistics*, 40(2):259–267, June.

Joakim Nivre et al. 2019. Universal dependencies 2.4. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*, pages 49–56, Boston, Massachusetts, USA, May 6 - May 7. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France, April.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.

Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 662–671, Denver, Colorado, May–June. Association for Computational Linguistics.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–117, Vancouver, Canada, July. Association for Computational Linguistics.

Corentin Ribeyre, Eric Villemonte de la Clergerie, and Djamé Seddah. 2014. Alpage: Transition-based semantic graph parsing with syntactic features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 97–103, Dublin, Ireland, August. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia, October. Association for Computational Linguistics.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 753–760, Manchester, UK, August. Coling 2008 Organizing Committee.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark, September. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018. An investigation of the interactions between pre-trained word embeddings, character models and POS tags in dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720, Brussels, Belgium, October-November. Association for Computational Linguistics.

Drahomíra Spoustová and Miroslav Spousta. 2010. Dependency parsing as a sequence labeling task. *The Prague Bulletin of Mathematical Linguistics*, 94(2010):7–14.

Miloš Stanojević and Raquel G. Alhama. 2017. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1666–1676, Copenhagen, Denmark, September. Association for Computational Linguistics.

Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium, October. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable dependency parsing as sequence labeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 717–723, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 187–197, Berlin, Germany, August. Association for Computational Linguistics.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. *IJCAI*, pages 1562–1567.

Alper Tokgöz and Gülşen Eryiğit. 2015. Transition-based dependency DAG parsing using dynamic oracles. In *Proceedings of the ACL-IJCNLP 2015 Student Research Workshop*, pages 22–27, Beijing, China, July. Association for Computational Linguistics.

Robert Vacareanu, George Caique Gouveia Barbosa, Marco A. Valenzuela-Escárcega, and Mihai Surdeanu. 2020. Parsing as tagging. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5225–5231, Marseille, France, May. European Language Resources Association.

Yannick Versley. 2014. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53, Dublin, Ireland, August. Dublin City University.

David Vilares and Carlos Gómez-Rodríguez. 2018. Transition-based parsing with lighter feed-forward networks. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 162–172, Brussels, Belgium, November. Association for Computational Linguistics.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2773–2781, Cambridge, MA, USA. MIT Press.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A transition-based algorithm for AMR parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado, May–June. Association for Computational Linguistics.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, Nancy, France, April.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia, July. Association for Computational Linguistics.

Anssi Yli-Jyrä and Carlos Gómez-Rodríguez. 2017. Generic axiomatization of families of noncrossing graphs in dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1745–1755, Vancouver, Canada, July. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 421–431, Berlin, Germany, August. Association for Computational Linguistics.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark, September. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August. Association for Computational Linguistics.

A Label sizes

Treebank	Encoding	Task 1 #labels (subsequence of transitions)	Task 2 #labels (dependency relations)	Treebank	Encoding	Task 1 #labels (subsequence of transitions)	Task 2 #labels (dependency relations)
Ancient Greek	rel-pos	166	27	Russian	rel-pos	169	44
	bracketing	210	27		bracketing	93	44
	arc-standard	46	27		arc-standard	65	44
	arc-eager	501	27		arc-eager	385	44
	arc-hybrid	41	27		arc-hybrid	45	44
	covington	3651	27		covington	1573	44
Chinese	rel-pos	201	45	Tamil	rel-pos	68	31
	bracketing	104	45		bracketing	49	31
	arc-standard	56	45		arc-standard	22	31
	arc-eager	817	45		arc-eager	79	31
	arc-hybrid	54	45		arc-hybrid	25	31
	covington	4846	45		covington	543	31
English	rel-pos	202	51	Uyghur	rel-pos	102	44
	bracketing	109	51		bracketing	58	44
	arc-standard	90	51		arc-standard	20	44
	arc-eager	509	51		arc-eager	185	44
	arc-hybrid	52	51		arc-hybrid	26	44
	covington	2804	51		covington	1459	44
Finnish	rel-pos	220	47	Wolof	rel-pos	102	40
	bracketing	107	47		bracketing	69	40
	arc-standard	77	47		arc-standard	69	40
	arc-eager	356	47		arc-eager	373	40
	arc-hybrid	46	47		arc-hybrid	46	40
	covington	2185	47		covington	920	40
Hebrew	rel-pos	162	40				
	bracketing	107	40				
	arc-standard	76	40				
	arc-hybrid	58	40				
	covington	2070	40				

Table 8: Number of labels for the transition-based parsing algorithms (arc-standard, arc-eager, arc-hybrid and Covington). We include two other existing encodings coming from different families (unrelated to transition-based parsing): the rel-PoS and bracketing encodings (Strzyz et al., 2019).

In Table 8 we show the number of distinct labels that several transition-based algorithms generate when mapped to a sequence labeling setup, according to our theory, on the UD treebanks used in our experiments. For comparison purposes, we also include the number of labels of other existing encodings for sequence labeling dependency parsing presented in Strzyz et al. (2019): the rel-PoS encoding (Spoustová and Spousta, 2010) and the bracketing encoding (Yli-Jyrä and Gómez-Rodríguez, 2017).

B Model parameters

BiLSTMs (NCRF++) All models were trained with learning rate 0.02 and learning rate decay of 0.05, for 100 epochs with batch size 8 for training and 128 for testing. The dimension of the pretrained word vectors was 100, and 30 for character embeddings. If PoS tag embeddings were used, their dimension was 25. The word/character hidden vector dimension was set to 800 and 50, respectively. We used 2 BiLSTM layers with momentum 0.9. Dependency labels were learned in a multitask learning setup.

BERT All models were fine-tuned with the learning rate 10^{-5} for 45 epochs with batch size 8. The maximum sequence length was set to 400, except for Russian, with 510.

C Postprocessing

In case a model outputs a label with an illegal action (due to violating preconditions in the transition system, e.g. a left arc in the arc-eager algorithm when the topmost stack node already has a head), we discard it and move to the next predicted action in the sequence.

The transition systems we implement guarantee acyclicity and single-headedness, but do not guarantee the output to have a single root as required in UD, so we perform simple postprocessing for this purpose. In the implementation of the transition systems, we distinguish between nodes that have not been assigned a head and nodes that have been assigned the dummy root, 0, explicitly as head. With this, it can happen

that a tree is generated with no roots (due to headless nodes). In this case all words with the predicted dependency relation "root" are assigned 0 as head, i.e. taken as candidates for being the true syntactic root. If still no root has been found, we choose the first token. If there are multiple roots, we pick the first token among them as root and link the others as dependents. Finally, any remaining headless nodes are assigned as dependents of the syntactic root (i.e. the node whose head is the dummy root).

D Encodings' upper bound on gold datasets

Table 9 reports the performance (UAS and LAS) for the nine treebanks when our BiLSTM models are trained with gold segmentation, tokenization and PoS tags, which serves as an empirical upper bound. In addition, the percentage increase when compared with the predicted setup (using UDpipe) is shown in parentheses. For orientation purposes, we provide the accuracy of UDpipe 1 on the UDv2.4 test sets in Table 10; for segmentation, tokenization and PoS tagging.

	Ancient Greek		Chinese		English	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	74.8 *(+7.6%)*	68.3 *(+13.9%)*	84.9 *(+31.8%)*	82.0 *(+36.7%)*	89.3 *(+8.9%)*	86.9 *(+10.7%)*
Brackets	69.1 *(+6.4%)*	63.1 *(+12.3%)*	84.2 *(+31.1%)*	81.2 *(+35.8%)*	88.8 *(+8.7%)*	86.5 *(+10.4%)*
Standard	65.1 *(+6.7%)*	59.6 *(+13.4%)*	82.8 *(+31.6%)*	79.9 *(+36.6%)*	85.7 *(+8.2%)*	83.4 *(+10.0%)*
Eager	64.3 *(+6.4%)*	58.6 *(+12.6%)*	83.7 *(+32.4%)*	81.1 *(+37.4%)*	86.6 *(+8.0%)*	84.2 *(+9.8%)*
Hybrid	66.2 *(+6.6%)*	60.3 *(+12.7%)*	83.0 *(+31.4%)*	80.1 *(+36.2%)*	86.5 *(+8.8%)*	84.2 *(+10.7%)*
Covington	67.9 *(+6.6%)*	61.6 *(+13.2%)*	76.3 *(+30.8%)*	73.4 *(+35.4%)*	85.1 *(+8.5%)*	82.8 *(+10.4%)*
avg	*(+6.7%)*	*(+13.0%)*	***(+31.5%)***	***(+36.4%)***	*(+8.5%)*	*(+10.3%)*
	Finnish		Hebrew		Russian	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	85.1 *(+6.2%)*	81.3 *(+8.5%)*	88.2 *(+40.1%)*	85.4 *(+43.7%)*	86.3 *(+4.3%)*	81.5 *(+5.3%)*
Brackets	86.1 *(+6.2%)*	82.5 *(+8.5%)*	87.5 *(+40.9%)*	84.7 *(+44.3%)*	86.0 *(+3.7%)*	81.6 *(+4.8%)*
Standard	83.1 *(+5.9%)*	79.6 *(+8.5%)*	84.2 *(+39.7%)*	81.4 *(+43.9%)*	82.2 *(+3.9%)*	77.9 *(+4.9%)*
Eager	83.9 *(+6.2%)*	80.0 *(+8.6%)*	85.6 *(+39.8%)*	82.6 *(+43.6%)*	83.7 *(+4.1%)*	79.2 *(+5.2%)*
Hybrid	83.5 *(+5.8%)*	80.1 *(+8.2%)*	85.1 *(+40.9%)*	82.2 *(+44.8%)*	82.4 *(+4.0%)*	78.1 *(+5.0%)*
Covington	83.5 *(+5.8%)*	79.7 *(+8.1%)*	84.0 *(+39.5%)*	81.1 *(+43.3%)*	82.2 *(+3.8%)*	77.8 *(+5.1%)*
avg	*(+6.0%)*	*(+8.4%)*	***(+40.2%)***	***(+43.9%)***	*(+4.0%)*	*(+5.0%)*
	Tamil		Uyghur		Wolof	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	76.5 *(+31.9%)*	68.7 *(+37.9%)*	74.3 *(+4.0%)*	61.1 *(+5.7%)*	86.0 *(+13.0%)*	81.8 *(+17.5%)*
Brackets	73.6 *(+28.5%)*	66.8 *(+36.3%)*	74.7 *(+3.7%)*	61.7 *(+5.5%)*	85.3 *(+13.1%)*	80.7 *(+17.2%)*
Standard	73.3 *(+28.2%)*	66.7 *(+37.2%)*	74.4 *(+4.0%)*	61.7 *(+6.0%)*	80.6 *(+11.9%)*	76.5 *(+16.6%)*
Eager	74.6 *(+32.8%)*	67.6 *(+40.1%)*	74.3 *(+3.3%)*	61.3 *(+4.3%)*	82.5 *(+12.2%)*	78.1 *(+16.4%)*
Hybrid	72.8 *(+27.7%)*	66.0 *(+35.5%)*	73.6 *(+3.2%)*	61.5 *(+5.0%)*	80.8 *(+11.9%)*	76.5 *(+16.5%)*
Covington	66.6 *(+27.4%)*	59.9 *(+33.8%)*	68.7 *(+3.5%)*	56.5 *(+5.7%)*	78.1 *(+11.7%)*	74.2 *(+x%)*
avg	***(+29.4%)***	***(+36.8%)***	*(+3.6%)*	*(+5.4%)*	*(+12.3%)*	*(+16.7%)*

Table 9: Performance of the models trained with gold segmentation, tokenization and PoS tags on the test sets, reported in UAS and LAS, with the percentage increase with respect to the results obtained with the predicted setup. The treebanks with the three highest averaged % of improvement in bold.

Language	Words (%)	Sentences (%)	UPoS (%)
Ancient Greek	100.0	98.9	82.2
Chinese	90.0	99.1	84.0
English	99.0	76.3	93.5
Finnish	99.7	88.6	94.3
Hebrew	85.0	99.4	80.5
Russian	99.5	96.2	95.0
Tamil	94.5	97.5	81.3
Uyghur	99.7	82.9	87.9
Wolof	99.2	92.0	91.7

Table 10: Prediction accuracy of UDpipe 1 on UDv2.4 treebanks.