Revisiting Supertagging and Parsing: How to Use Supertags in Transition-Based Parsing

Wonchang Chung
Dept. of Computer Science
Columbia University
New York, NY, USA
wc2550@columbia.edu

Siddhesh Suhas Mhatre
Dept. of Computer Science
Columbia University
New York, NY, USA
sm4083@columbia.edu

Alexis Nasr
LIF
Université Aix Marseille
Marseille, France
Alexis.Nasr@lif.univ-mrs.fr

Owen Rambow
CCLS
Columbia University
New York, NY, USA
rambow@ccls.columbia.edu

Srinivas Bangalore
Interactions, Inc.
New Providence, NJ, USA
sbangalore@interactions.com

Abstract

We discuss the use of supertags derived from a TAG in transition-based parsing. We show some initial experimental results which suggest that using a representation of a supertag in terms of its structural and linguistic dimensions outperforms the use of atomic supertags.

1 Introduction

The notion of supertagging was introduced by Bangalore and Joshi (1999). A supertag is the name of an elementary tree assigned to a word in a TAG derivation of the sentence. A supertag therefore encodes not only the part of speech, but also the syntactic properties of the word. They proposed a two-step approach to parsing: a supertagger determines the supertag for each word in a sentence, and a deterministic and rule-based "lightweight dependency analyzer" then derives the structure from the supertags.

The MICA parser (Bangalore et al., 2009) uses a supertagger and a subsequent chart parser which takes the 10-best supertags for each word as input. MICA uses a probabilistic context free grammar which is lexicalized on the supertags, but not on words. The MICA parser is fast, and has good performance. Nasr and Rambow (2006) showed that the MICA approach outperforms the lightweight dependency analyzer of Bangalore and Joshi (1999). To our knowledge, MICA is the only TAG parser trained on the Penn Treebank that uses supertagging; it is freely available.[1]

The MICA parser has several drawbacks: while it is fast, the time complexity is $O(n^3)$. Furthermore, the system is complex as the chart parser itself is compiled using the SYNTAX system (Boullier and Deschamp, 1988), making further development difficult. Finally, it is unclear how to include recent advances in lexical representation (word embeddings) and machine learning (deep learning).

This paper presents a new parser based on TAG, which uses supertagging and a distinct parsing step. Unlike MICA, the parsing is based on the transition-based parser of Nivre et al. (2004). While there has been some work using supertags with transition-based parsing (Ouchi et al., 2014), this is the only work (to our knowledge) which specifically refers to TAG grammar.

Bangalore et al. (2009) train a version of MALT with gold and predicted supertags. MALT can exploit the gold supertags, but not the predicted supertags (they do not improve over not using them). The problem with using supertags in transition-based parsing is that exploiting n-best supertag input is difficult, and given the large number of supertags, supertagging is hard and the 1-best supertag is not good enough to allow for a good parse to be constructed. In this paper, we present initial investigations to address this problem. We decompose the supertag into linguistic dimensions, which provides for a generalization of the notion of supertag.

2 Corpus and Grammar

We use the grammar and the corpus extracted by Chen (2001). This grammar was engineered in such a way that the derivation trees are meaningful deep-syntactic representations. This grammar was also used in the MICA parser (Bangalore et al., 2009). It has 4725 elementary trees extracted from

[1] urlhttp://mica.lif.univ-mrs.fr

Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12), pages 85–92,
Düsseldorf, Germany, June 29 - July 1, 2016.

the training set of the WSJ portion of the Penn Treebank (Sections 01-22). Every sentence in the corpus is given a derivation. Sentences in the development set (Section 00) and the test set (Section 23) may contain elementary trees that have not been seen in the training corpus.

We automatically analyzed the elementary trees that make up the extracted TAG, assigning each tree a vector of 20 dimensions. These dimensions fall into three categories:

- Dimensions that describe the phrase structure of the elementary tree. We concentrate on aspects that we think will be important for parsing.
- Interpretations of the tree. These are linguistic dimensions which abstract from the phrase structure of the tree.
- Linguistic transformations on the tree. These are syntactic variations that the tree encodes, such as *wh*-movement.

This approach of breaking down a supertag into components is inspired by the hypertags of Kinyon (2000). Our set of dimensions is shown in Figure 1.

3 Supertagging

The supertagger architecture is very simple: supertags are predicted independently of each other. The prediction is performed using an online passive-agressive algorithm (Crammer et al., 2006). We used the implementation of the Python Scikit-Learn library.[2]

The classifier uses a total of 26 features: the word to be supertagged, its part-of-speech tag as well as the 6 preceding and following words and part-of-speech tags. To vectorize the feature data, the one-hot encoding method was used.

Training was performed on the training set of the WSJ portion of the Penn Treebank (950,028 tokens) and the evaluation on the development set (40,461 tokens). In order to reduce the amount of memory used for training, the sparse matrix constructor was used. The peak amount of memory used for training task was less than 50GB, and the processing time was less than 1 hour in wall-clock time on our machines.

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html`

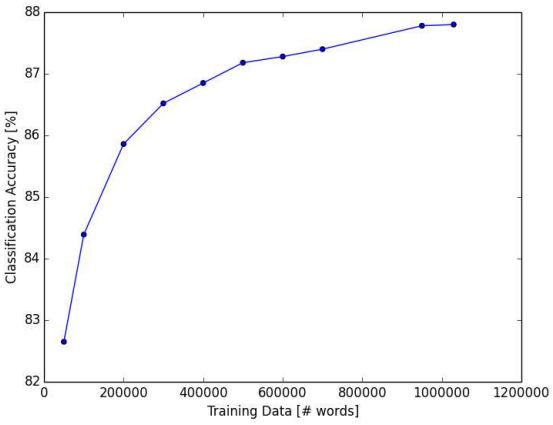

Figure 2: Learning curve of the supertagger

The supertagger accuracy is 87.88% on the development set, a bit lower than the results obtained on this data by Bangalore et al. (2005), which was 88.53%. The learning curve is shown in Figure 2.

4 Parsing

4.1 Background

The parser used in this study, named SuTra (for Supertag- and Transition-based Parser), is a standard transition based parser (Nivre et al., 2004). Giving a thorough descriptions of transition based parsers is not the aim of this paper; we will just briefly describe below the basic ideas behind transition based parsing to allow the reader to follow the rest of the paper.

Transition-based parsers are based on two fundamental objects: configurations and transitions

A configuration (s, b, D) describes the state of the parsing process at a given time. b is a buffer that contains the words of the sentence to parse not yet processed. The leftmost word of the buffer is noted b_0. b_0 can be taken from the buffer and pushed on the stack s. D is a set containing dependencies that have been built to this point by the parser. The parser tries to build a dependency between the word that is on the top of the stack (s_0) and the next word in the buffer (b_0). Two types of attachements are considered, left attachements that have as a governor word b_0 and as a dependent s_0 and right attachements that have s_0 as governor and b_0 as dependent. The initial configuration of the parser is $([], [w_1 \dots w_n], \emptyset)$: the stack is empty, the buffer contains all the words of the sentence to parse and the dependency set is empty. A fi-

Dimension	Description
Dimensions describing the phrase structure of the elementary tree	
root	The label of the root node of the tree
lfront	A list of substitution nodes to the left of the lexical anchor; each node is listed with its category, its node type (substitution or co-head), and its deep-syntactic argument label
rfront	Same as lfront, but for the substitution nodes to the right of the lexical anchor
adjnodes	A list of nodes at which adjunction can occur
substnodes	A list of all substitution nodes of the tree
coanc	Does this tree have a co-anchor?
modif	For modifier auxiliary trees, the cateory of its root node (and thus of its foot node)
dir	The direction in which a modifier auxiliary tree adjoins
Dimensions interpreting the elementary tree	
predaux	Is the tree a predicative auxiliary tree (i.e., a tree used for matrix clauses)?
pred	Is this tree a nominal, adjectival, or prepositional tree which projects a predicative structure, i.e., takes a subject (even if not realized)?
appo	Is this tree an apposition?
comp	Does this tree have a complementizer (which is a co-head in this grammar), and if yes, on what side of the anchor?
dsubcat	Deep subcategorization frame for the anchor, listed in order of argument number (i.e., not necessarily surface order), with substitution node category and strongly governed category, if any
ssubcat	Surface subcategorization frame for the anchor, listed in order of argument number (i.e., not necessarily surface order), with substitution node category and strongly governed category, if any
particle	Does this tree contain a particle (POS tag RP)?
Dimensions describing linguistic transformations on the elementary tree	
voice	Voice for verbal trees
wh	Is there a *wh*-moved dependent?
rel	Is the tree a relative clause?
esubj	Is the subject of the tree empty?
datshift	For ditransitive trees, did dative shift happen?

Figure 1: Description of tree dimensions

nal configuration is a configuration for which the buffer is empty.

A transition operates on a configuration c_i to produce a configuration c_{i+1}. In our implementation, three types of transitions are defined:

Left Arc builds a dependency (b_0, l, s_0) (a transition that has b_0 as a governor, s_0 as a dependent and l as a label). This transition adds the new dependency to D and pops the stack.

Right Arc builds a dependency (s_0, l, b_0). This transition adds the new dependency to D and replaces b_0 with s_0.

Shift does not create a new dependency, it just remove b_0 from the buffer and pushes it on the stack.

The parser is a greedy deterministic parser. Given a configuration, it predicts the most likely transition to make. A new configuration is produced and the process iterates until a final configuration is reached. The dependency structure produced is the set D.

The heart of the parser is the classifier that predicts which transition to make given a configuration. The number of possible configurations being very large, we decompose a configuration into a feature vector. During training, the classifier associates a score to each feature. At decoding time, the classifier adds the scores of the features corresponding to the current configuration in order to select the most likely transition. The classifier used in this work is a simple averaged percep-

tron (Freund and Schapire, 1999).

The feature templates used by the classifier are of three sorts:

Word features describe different aspects of the words that are present either on the stack of the parser or in the buffer. They are of the form `(s|b)(0|1|2|3)(f|l|c|p|m)` where:

- `s|b` indicates whether the word described is in the stack or the buffer
- `0|1|2|3` indicates the position of the word in the stack or the buffer (`s0` is the top stack word and `b0` is the first word in the buffer).
- `f|l|c|p|m` indicates whether we are referring to the form of the word (`f`), its lemma (`l`), its coarse part of speech (`c`), its part of speech (`p`) or its morphological features (`m`).

Distance features indicate the distance in the string between two words. They are of the form `d_X_Y` where `X` and `Y` correspond to words either on the stack or in the buffer. The only feature of this category that is used is `d_s0_b0`.

Structural features describe some aspects of the dependency structure built so far by the parser. They are of three sorts:

- `l_X` which indicate the syntactic function (role label) of the leftmost dependent (if any) of word `X`. Two features of this category are defined: `r_s0r` and `r_b0r`.
- `r_X` which indicate the syntactic function (role label) of the rightmost dependent (if any) of word `X`. Two features of this category are defined: `l_s0r` and `l_b0r`.
- `n_X` which indicate the number of dependents of word `X`. Two features of this category are defined: `n_s0` and `n_b0`.

Configuration features describe some aspects of the current configuration of the parser. They are of four sorts:

- `sh` indicates the height of the stack
- `bh` indicates the number of elements in the buffer
- `dh` indicates the number of dependencies built so far
- `tn` with `n=1,2,3,4`, indicates the n^{th} preceding transition that led to the current configuration

Each feature template can be used independently or in combination with others, in which case a weight is computed for a combination of their values.

4.2 Parser 1: Baseline Parser without Supertags

We start by describing our baseline parser, which is SUTRA without any supertag features at all.

Table 4 shows the set of feature templates (called a feature model) used for our baseline parser. (All of the tables related to the machine learning features are at the end of the paper.) Feature templates 1 to 18 are simple feature templates, those ranging from 19 to 29 are combination of two simple feature templates.

The performance of the baseline parser is shown in Table 1 in the first row, with separate results for labeled attachment score (LAS) and unlabeled attachment score (UAS). Since we are not using supertags in this experiment, the results are the same for gold and predicted supertags.

For the sake of comparison, we also give results for a MALT parser trained on our corpus (2nd line in Table 1; the results are taken from (Bangalore et al., 2009)). Our baseline results are directly comparable to those for MALT without supertags, as both are transition-based parsers which do not use supertags. We see that our results are a little worse, which we attribute to differences in the machine learning, and differences in the feature set used. However, for the sake of the experiments in this paper, we take our results as meaning that we have replicated the previous results.

4.3 Parser 2: Using Supertags

We now use supertags. In the first experiment, we simply add the supertags as labels in our parser by means of the following word feature templates: `(s|b)(0|1|2|3)(s)`, where the first two components of the templates `(s|b)` and `(0|1|2|3)` keep the same meaning as before and `s` refers to the supertag of the word. The feature model of Parser 2 adds to the feature model of the baseline Parser the feature templates shown in Table 5. These templates correspond to templates of the baseline parser in which part of speech tags are remplaced by supertag tags.

Parser		Gold Stags		Predicted Stags		
		UAS	LAS	Stag acc.	UAS	LAS
Baseline	Words, POS tags	—	—	—	87.65	85.23
MALT	Words, POS tags	—	—	—	88.9	86.9
P2	Words, POS tags, stags	97.02	96.00	87.88	89.83	87.75
MALT-Stag	Words, POS tags, stags	97.20	96.90	88.52	88.50	86.80
MICA	Stags only	97.60	97.30	88.52	87.60	85.80
P3	Words, POS tags, stags, stag dimensions	97.46	96.51	87.88	89.96	87.86

Table 1: Results for different configurations.

The results of P2 are displayed in Table 1 in row 3. As one can see, when feeding the parser with gold supertags, the results accuracy of the parser jumps to 96.97 UAS and 95.99 LAS. Supertags carry much more syntactic information than just POS tags that the parser can make use of in order to predict the syntactic structure of the sentence. When supertags are predicted with the supertagger of section 3, the accuracy dips to 89.86 UAS and 87.75 LAS, respectively. This represents an absolute increase of 2.24 points of UAS and 2.52 points of LAS with respect to the baseline parser. We also compare P2 to MALT using supertags, shown in row 4. We see that our parser outperforms MALT with stags by a small margin when using predicted supertags (but not gold supertags). Part of the difference in the predicted supertags is due to the use of gold POS tags in our experiments, so we conclude that we are again replicating the previous result.

We also provide the results for MICA (row 5). We see that for gold supertags, MICA provides the best overall results, but not for predicted supertags. This is because MICA in fact *only* uses supertags.

4.4 Parser 3: Using Dimensions of Supertags

We now perform experiments to see whether the individual dimensions of the supertags can help in parsing. The motivation is that if a supertag is incorrectly predicted, some of the dimensions may still be correct (for example, the predicted supertag has a transitive verb instead of an intransitive verb, but the subject is empty in both supertags).

In order to be able to exploit the supertag dimensions in the parser, we add the following word feature templates: `(s|b)(0|1|2|3)(A|B|...|T|U)` where, as before, the first two components of the templates `(s|b)` and `(0|1|2|3)` keep the same signification and the letters `A` to `U` refer to one dimension of the vector representation of supertags. The correspondence is given in Table 3. The feature model of parser 3 is the union of the feature model of P2 and the features of Table 6.

We observe that we cannot use all dimensions of the linguistic vector representation of the supertag, because the combinations would result in a combinatorial explosion in the number of features for machine learning. In order to gain a better understanding of which dimensions of the decomposed supertags are useful for parsing, we performed ablation studies, first on the dimensions, and then on the machine learning features. We discuss them in turn.

In the first study we removed each dimension of the supertag (eg. dsubcat, ssubcat, ...) in turn and computed the parsing accuracy. For this ablation study, we use a feature model that comprises simple features derived from supertags and non-supertags. Specifically, this model comprises the following features: features 1 through 18 from Table 4, plus s0x, s1x, b0x, and b1x, where x is a variable denoting the dimensions (represented as in Table 3). We use this model because it is a simple model. The results are shown for gold and predicted stags in Table 2.

For the gold experiments (first two columns), we see that mainly the dimensions that describe the phrase structure are useful for parsing: all of these dimensions except for coanc help, and all the most useful dimensions are of this type. This is because in a TAG grammar, the phrase structure encodes exactly how trees can combine in the parse, so that this is the information needed for a correct parse. In addition, we have several of the dimensions relating to transformations that help a bit. When we look at the predicted supertags, we

see that seven of the eleven dimensions that are useful for the gold condition are still useful, most of them structural. However, we also expect to see a shift, as some dimensions are harder to predict with sufficient accuracy. In particular, we see that rfront no longer helps. We hypothesize that this is because there is a large number of possible values for this dimension (more than for lfront, because of the syntax of English), and that an error immediately reduces the usefulness of this dimension. Perhaps as a result, the dsubcat dimension is useful in the predicted condition. The dsubcat dimension abstracts over actual phrase structure, and therefore has a smaller set of possible values, while still providing some of the same information that the dsubcat dimension provides (what dependents this head expects).

Now we turn to the second ablation study, in which we concentrate on specific features rather than dimensions. To pick out those individual features (eg. s0A, s1B, ...) in the feature model of the remaining supertag dimensions (eg. dsubcat, ssubcat, ...) that cause a decrease in performance, we performed another level of feature ablation. We use the same feature model we used in the first ablation study. For each supertag dimension that remained after the first ablation study, we removed each corresponding machine learning feature one by one in and computed the parsing accuracy. For example, if we consider supertag dimension dsubcat (represented as A), then we performed experiments where we removed the feature s0A in the feature model, followed by s1A and so on, every time observing the effects on the parsing accuracy. This was done for every remaining supertag dimension. Again, at the end of this set of experiments we eliminated those features from our feature model that cause a decrease in parsing performance when included. Because of the large number of results, we do not present them in detail.

Till now we had considered only the supertag dimensions in our model independently. Our last set of experiments comprises combining some of these features from the feature model. Here, we used our intuition to propose certain combinations. One set of features we combined were the ones corresponding to the dimensions 'lfront' and 'rfront'. These correspond to ordered list of frontier nodes to the left and right of the main lexical anchor respectively. We merge both lfront or rfront with the root of the elements on top of the stack and buffer. These features were then combined with the 'dir' dimension which gave encouraging results. We tried 8 different combinations in this manner and got the best results for the following feature model which is shown in Table 6. This set of features also includes those features that correspond to non-supertag features that gave us the best performance.

The results of P3 are displayed in the last line of Table 1. As one can see the decomposed representation of the supertag has a beneficial impact for both the gold and predicted supertag conditions. The error reduction for gold supertags (UAS) is 15%; for predicted, the error reduction is much smaller (1%). We think this smaller error reduction may be due to the fact that in our feature engineering, which was guided by our intuition, we did not take into account the accuracy of different dimensions, assuming implicitly the case in which the dimensions are correctly predicted. Our new results are better than the best published parsing results so far on this corpus, as far as we know.

5 Conclusion

We have presented work in progress, that shows that supertagging can be useful for transition-based parsing. Our initial experiments suggest that considering the dimensions of the supertag can help further.

A major problem is devising machine learning features for the parser from the dimensions, given the very large number of possibilities due to the combinatorics of the combined features. In future work, we plan to use deep learning to obviate the need for feature engineering. This will also entail using word embeddings, which we will also use for supertagging. We will look to the rich literature on supertagging and parsing in CCG for guidance. In addition, we will also start using predicted POS tags in our experiments.

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

Srinivas Bangalore, Patrick Haffner, and Gaël Emami. 2005. Factoring global inference by enriching local representations. Technical report, AT&T Labs – Reserach.

	Gold		Predicted	
Dim.	LAS	UAS	LAS	UAS
predaux	+.07	+.05	+.04	0
rel	+.07	+.06	**-.06**	**-.05**
particle	+.05	+.05	**0**	**-.05**
coanc	+.04	+.02	**0**	**-.07**
ssubcat	+.04	+.03	0	-.01
wh	+.02	+.02	+.05	+.02
comp	+.01	+.01	**-.01**	**-.05**
dsubcat	+.01	+.02	**-.07**	**-.09**
pred	0	0	+.06	+.03
none	94.87	95.83	83.15	85.43
datshift	**-.01**	**-.01**	+.07	+.03
voice	**-.02**	**-.02**	**-.03**	**-.07**
esubj	**-.03**	**-.02**	**-.03**	**-.06**
appo	**-.04**	**-.04**	+.02	0
substnodes	**-.04**	**-.02**	0	-.01
adjnodes	**-.04**	**-.04**	**-.17**	**-.19**
modif	**-.07**	**-.10**	**-.09**	**-.11**
lfront	**-.07**	**-.06**	**-.05**	**-.07**
rfront	**-.16**	**-.15**	+.02	-.02
root	**-.32**	**-.32**	**-.09**	**-.12**
dir	**-.90**	**-.88**	**-.40**	**-.43**

Table 2: Ablation study for supertag linguistic features, with gold standard supertags and predicted supertags. Each row lists one feature which was removed in turn. The resulting difference in performance is shown (labeled and unlabeled dependency accuracy without punctuation), first for gold supertags, then for predicted supertags. If a result gets worse upon removal of a feature (negative value), then that dimension is important. We show the retained dimensions by boldfacing their resulting change in accuracy.

Srinivas Bangalore, Pierre Boullier, Alexis Nasr, Owen Rambow, and Benoît Sagot. 2009. MICA: A probabilistic dependency parser based on tree insertion grammars. In *NAACL HLT 2009 (Short Papers)*.

Pierre Boullier and Philippe Deschamp. 1988. Le système SYNTAX™ – manuel d'utilisation et de mise en œuvre sous UNIX™. http://syntax.gforge.inria.fr/syntax3.8-manual.pdf.

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

Yoav Freund and Robert E Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Alexandra Kinyon. 2000. Hypertags. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*.

Alexis Nasr and Owen Rambow. 2006. Parsing with lexicalized probabilistic recursive transition networks. In *Finite-State Methods and Natural Language Processing*, Springer Verlag Lecture Notes in Commputer Science.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56, Boston, Massachusetts, USA.

Hiroki Ouchi, Kevin Duh, and Yuji Matsumoto. 2014. Improving dependency parsers with supertags. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 154–158, Gothenburg, Sweden, April. Association for Computational Linguistics.

A	dsubcat	B	ssubcat	C	voice
D	comp	E	datshift	F	root
G	lfront	H	rfront		
J	adjnodes	K	substnodes	L	rel
M	particle	N	coanc	O	modif
P	dir	Q	pred	R	esubj
S	wh	T	appo	U	predaux

Table 3: List of supertag dimensions used, with short name used in the tables of machine learning features

1	s0c								
2	s0f								
3	s0p								
4	s1p								
5	b0c								
6	b0f								
7	b0p								
8	b1c								
9	b1f								
10	b1p								
11	b2p								
12	b3p								
13	l_s0r								
14	r_s0r								

15	l_b0r								
16	r_b0r								
17	n_s0								
18	n_b0								
19	s0c	b0c							
20	s0f	b0f							
21	s0p	b0p							
22	b0c	b0f							
23	b0p	b0f							
24	b0p	l_b0r							
25	s1c	b1c							
26	s1p	b1p							
27	b1c	b2c							
28	b1p	b2p							

29	s0c	b0c	b0f						
30	s0c	s0f	b0c						
31	s0p	b0p	b0f						
32	s0p	b0p	b1p						
33	s0p	l_s0r	r_s0r						
34	s0p	s0f	b0p						
35	s0c	b0c	d_s0_b0						
36	s0p	b0p	d_s0_b0						
37	s1p	s0p	b0p						
38	b0p	b1p	b2p						
39	b1c	b2c	b3c						
40	b1p	b2p	b3p						
41	b1c	b1f	b2c	b3c					
42	b1p	b1f	b2p	b3p					
43	b1c	b1f	b2c	b2f	b3c				
44	b1p	b1f	b2p	b2f	b3p				

Table 4: Baseline feature model

45	s0s								
46	s1s								
47	b0s								
48	b1s								
49	b2s								
50	b3s								
51	b0s	b0f							
52	b0s	l_b0r							
53	b1s	b2s							
54	s0s	b0s							
55	s1s	b1s							
56	s0s	b0s	b0f						
57	s0s	b0s	b1s						
58	s0s	b0s	d_s0_b0						
59	s0s	l_s0r	r_s0r						
60	s0s	s0f	b0s						
61	s0s	s1s	b0s						
62	b0s	b1s	b2s						
63	b1s	b2s	b3s						
64	b1s	b1f	b2s	b3s					
65	b1s	b1f	b2s	b2f	b3s				

Table 5: Parser 2 feature model (in addition to the features shown in Table 4).

38	s0A	47	b0F	57	s0J	66	s1P		
39	b1A	48	b1F	58	b0J	67	b0P		
40	s0B	49	b0G	59	s0K	68	b1P		
41	b1B	50	s0H	60	b1K	69	s0Q		
42	s0C	51	s1H	61	s0M	70	s1Q		
43	b0C	52	b0H	62	b0M	71	b0R		
44	b0D	53	b1H	63	s0O	72	b0S		
45	s1E	55	b0I	64	b1O	73	s0U		
46	s1F	56	b1I	65	s0P				

72	s0H	b0F							
73	s0F	b0G							
74	b0H	b1F							
75	b0F	b1G							
76	s0H	s1F							
77	s0F	s1G							
78	s0H	b0F	s0P						
79	s0F	b0G	b0P						
80	b0H	b1F	b0P						
81	b0F	b1G	b1P						
82	s0H	s1F	s0P						
83	s0F	s1G	b1P						
84	s1H	s0F	b0G						
85	s0H	b0F	b1F						
86	s0H	b0F	d_s0_b0						
87	s0F	b0G	d_s0_b0						
88	s0H	b0F	s0P	d_s0_b0					
89	s0F	b0G	b0P	d_s0_b0					

Table 6: Parser 3 feature model (in addition to the features shown in Table 4 and Table 5).