Keeping it Simple: Generating Phrase Structure Trees from a Hindi Dependency Treebank

Himanshu Yadav[†], Ashwini Vaidya[‡] and Samar Husain[‡]

[†]Jawaharlal Nehru University, India
[‡]Indian Institute of Technology Delhi, India
yadavhimanshu059@gmail.com,
ashwini.vaidya@gmail.com, samar@hss.iitd.ac.in

Abstract

Converting a treebank from one representation type to another poses several challenges [4] [3]. These challenges are contingent on (amongst other things) the information encoded in source representation and the information required in target representation. In this paper, we propose a conversion algorithm that converts the Hindi-Urdu Dependency Treebank (HUTB) to a Phrase Structure (PS) representation. In order to do this, we extract structural information (for projecting heads) as well as predicate-argument information from the dependency structure (DS). The resulting PS trees are relatively flat with few empty categories (ECs) and are very close to the DS trees in their syntactic content. Our algorithm generates 'valid' PS trees. The validity is based on certain metrics such as well-formedness, linearity, etc. These trees could be further transformed for theory specific analyses.

1 Introduction

Most modern treebanks make use of either Dependency Structure (DS) or Phrase Structure (PS) representation to encode syntactic phenomenon. DS representation has been used more frequently for free word order languages like Czech, Hindi, Turkish, Russian etc. In addition, due to the development of efficient dependency parsing algorithms, use of dependency formalism for treebanking has become common [10]. While DS representation offers several advantages, it is also known that certain types of structural constraints (e.g. c-command) that are based on constituent structures cannot be formalized in dependency structure. Such constraints are known to play an important role in accounting for various syntactic phenomena (e.g. reference resolution, syntactic islands, etc.) [11] (also see [6, 8]). In addition, the PS representation lends itself naturally to capture the predictive nature of human sentence processing via left-corner phrase structure parsing algorithm [7]. It is therefore important to explore both these formalisms in order to investigate

their usefulness in language modeling and in building various applications. Since a treebanking task uses a single representation, one needs to transform the original representation to the desired representation.

The complexity of a conversion algorithm will be contingent on (a) the information encoded in the source vs the information required in the target, (b) word order variation in the language, and (c) the choice of the target representation. These factors can lead to small or large divergence between the source and target representation. For example, Bhatt and Xia [4] required an intermediate layer in order to handle the information asymmetry between the source and target representation. Similarly, following Bhatt and Xia [4], Luu et al. [9] also introduced an additional intermediate layer to handle projectivity in their target PS. In this work, we propose a conversion algorithm[1] that converts the DS tree in HUTB to the PS representation with the aim of maintaining 'content of representation' [12] between the source and target. The resulting trees are relatively flat with few empty categories (ECs), i.e. these PS trees are very close to the DS trees as far as syntactic information is concerned.

Our paper is organized as follows: Section 2 discusses the salient properties of the PS representation scheme that is generated by the algorithm. In Section 3, we describe our conversion algorithm. Following this in section 4 we evaluate its performance. We conclude the paper in section 5 and discuss future directions.

2 The PS representation scheme

Our proposed PS representation is relatively flat as it does not encode syntactic roles (like Subject, Object etc.) structurally. The PS trees also do not necessarily have binary branching or syntactically driven ECs. The flatness of the trees is a by-product of the following representation choices (a) No intermediate projection for heads, (b) No intermediate projection for dependents, (c) No binary branching. These choices, in fact, relate to the questions pertaining to head projection and dependent attachment raised by Xia and Palmer [14]. They point out three issues to be addressed by any conversion algorithm - (1) What kind of projection can a category (X) have? (2) How far should a dependent project before it attaches to its head? (3) To what position on a head's projection chain[2], should a dependent's projection attach? Regarding (1) and (2), as stated earlier, we do not have intermediate projection for any head or dependent. Regarding (3), in our phrase structure, a dependent is always a sister of its head. This is illustrated in Figure 1.

In addition, we do not have any structurally distinguished position for encoding syntactic roles. We capture this information using function tags like '–SUBJ' for Subject, '-OBJ-1' for direct object etc. These design considerations are motivated

[1]The implemented algorithm can be downloaded from `https://github.com/yadavhimanshu059/Hindi_DS_to_PS_Conversion`

[2]A chain corresponds to all the intermediate projections of a category (X) upto its maximal projection.

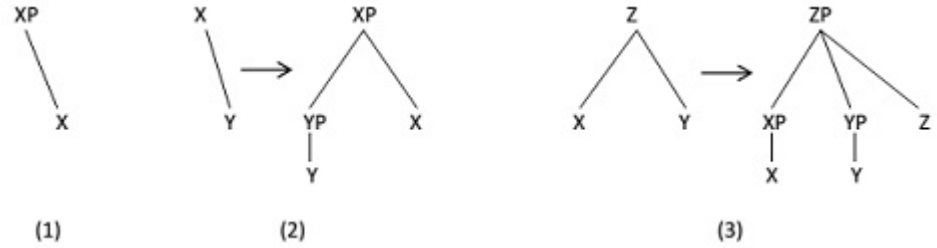

Figure 1: The choices taken in our algorithm regarding three questions raised by Xia and Palmer [14]

by the aim of maintaining 'content of representation' [12] between the source and target.

Interestingly, Collins et al. [5] have also argued for flat PS representation scheme for Czech, which is also a free word order language like Hindi. On the other hand, PS representation scheme for Hindi by Bhatt et al. [2] maintains fixed structural positions for core arguments etc. leading to multiple empty categories. They consistently assume a binary branching representation while maintaining a linear word order. It is easy to see that getting to a representation proposed by [2] is difficult because of the information asymmetry and the divergent analyses of various phenomena in DS and PS [3]. Of course, we are not claiming that a more detailed PS scheme for Hindi (e.g. Bhatt et al. [2]) will not be useful at all. Indeed, from a theoretical perspective, PS trees with elaborate empty categories might be very useful. In fact, we envisage our PS schema (that is closer to DS) to be a useful resource in order to reach such a representation.

3 DS to PS conversion algorithm

As stated earlier, our conversion algorithm does not use any additional information other than the DS. We extract modifier-modified information (for projecting heads) as well as predicate-argument information from the DS.

We illustrate our conversion algorithm with the example in Figure 2 below. The algorithm applies the following rules (i) rules for head-projection (ii) rules for argument-adjunct distinction (iii) rules for joining argument structure sub-trees (iv) rules for joining clauses in case of multi-clause constructions.

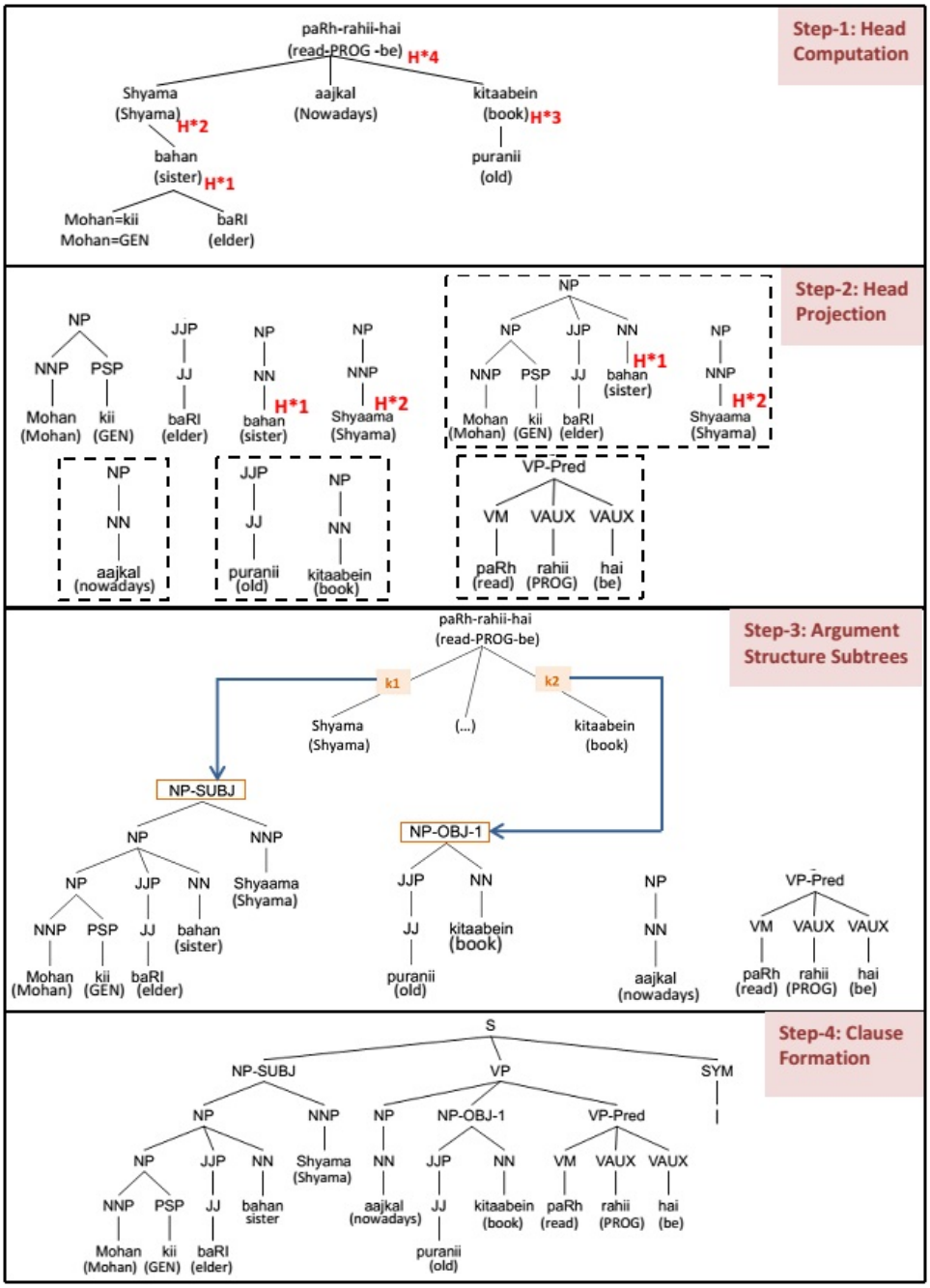

Figure 2: DS to PS conversion steps for example 1. In Step-1, H*1, H*2 etc. are the computed heads from the DS. In Step-2, these heads are projected recursively to form phrase fragments. Dashed boxes show a set of phrase fragments. In Step-3, we extract dependency relation information from DS and use the set of phrase fragments from Step-2 as input to form argument structure subtrees. These subtrees are combined in Step-4 to get the clausal PS tree.

Illustrative Example (see Figure 2)

(1) Mohan=kii baRI bahan shyaama aajkal
Mohan.N.m.sg=Gen elder.adj sister.N.M.sg Shyama.N.f.sg nowadays.adv
puranii kitaabein paRh-rahii-hai
old.adj.f.pl book.f.pl read-prog.f.sg-be
'Mohan's elder sister Shyama is reading old books these days'

We use the sentence shown in Example 1 to illustrate our conversion process. In step-1 (see Figure 2), we use the modifier-modified relation from the DS tree to compute the heads of phrases in the corresponding PS tree. The modified element takes the head position in phrases. The heads are marked with H* in Figure 2. In step-2, the heads extracted from DS are recursively projected to generate phrase-fragments. We use the part-of-speech (POS) tag information from DS at this level. Thus, a head with POS tag X projects to XP in phrase structure. The modifiers of a head in DS take the non-head positions in a minimal phrase fragment.

In step-3, we use the dependency relation label information of the modifiers of the predicate in DS in order to make the distinction between arguments and adjuncts. The following dependency relation to syntactic role mappings were used: k1 – subject, k2/k2p – object, etc. Previous attempts to extract argument information from the DS have shown that this mapping can correspond to predicate-argument information [1, 13]. Once the argument-adjunct distinction has been made, we then create sub-trees by adjoining the phrase fragments corresponding to NP-SUBJ, NP-OBJ etc. This means that syntactic roles such as subject and object are not encoded structurally; rather this information is encoded through the phrase tag. We get the complete PS tree by joining sub-trees as shown in step-4.

If a sentence is composed of two or more clauses, each clause is formed using the same procedure shown as in Figure 2. We then adjoin these clauses at a clause joining step.

3.1 Basic Argument Structure

Dependency Label	Phrase-Tag
k1	-SUBJ
k2, k2p	-OBJ-1
k4, k2g	-OBJ-2
k4a	-SUBJ-Dat
pk1	-SUBJ
jk1	-J-SUBJ
k2s	-OBJ-Comp

Table 1: Dependency label to Phrase-Tag mapping

As stated earlier, we use phrase tags instead of structural encoding for argument structure information. In a minimal clause, arguments licensed by verb are repre-

sented using phrase tags like -SUBJ, -OBJ-1, -OBJ-2 etc (see Table 1). It makes our resulting trees flatter and non-binary branched, see Figure 3. Previous attempts to extract argument information from the DS have shown that this mapping can correspond to predicate-argument information [1, 13].

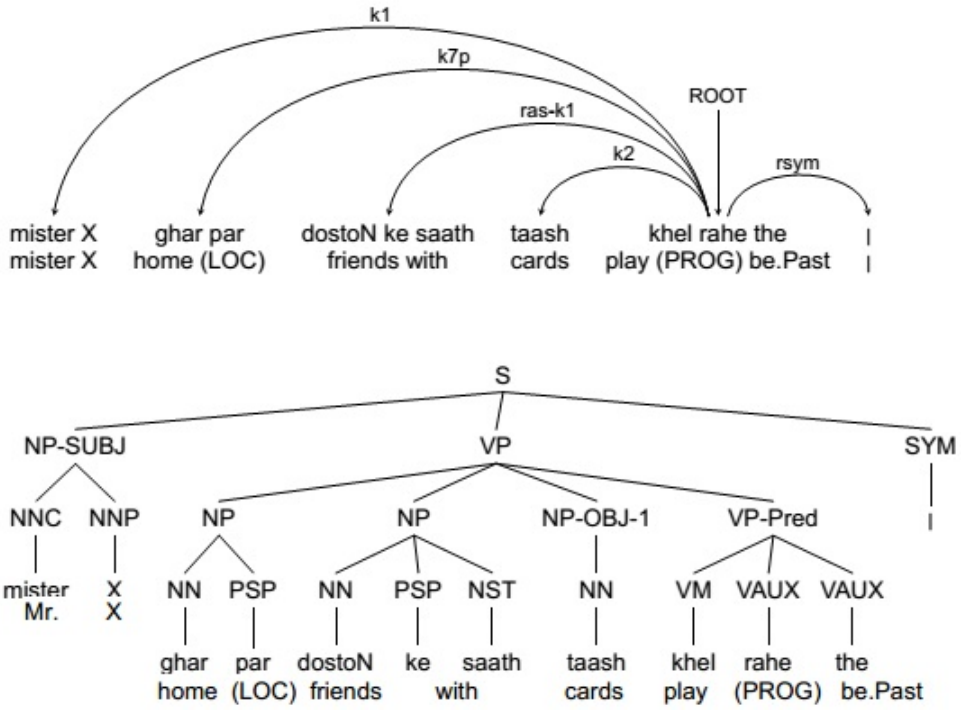

Figure 3: A DS and the derived PS showing basic argument structure for the sentence 'Mr. X was playing cards with friends at home.'

3.2 Non-finite Clause

The Hindi-Urdu Treebank (HUTB) makes a distinction between finite verbs and non-finite verbs. We maintain predicate-argument information in a minimal non-finite clause using phrase tags -SUBJ, -OBJ-1, -SUBJ-Dat etc. Although we do not posit any structural preferences for a non-finite clause, we encode a distinction between finite and non-finite clause using phrase names S and S-NF/S-NN respectively. An example non-finite clause is shown in Figure 4. Our system does not add empty categories for shared subjects. If there is any empty node in DS, then, we maintain it in resulting PS.

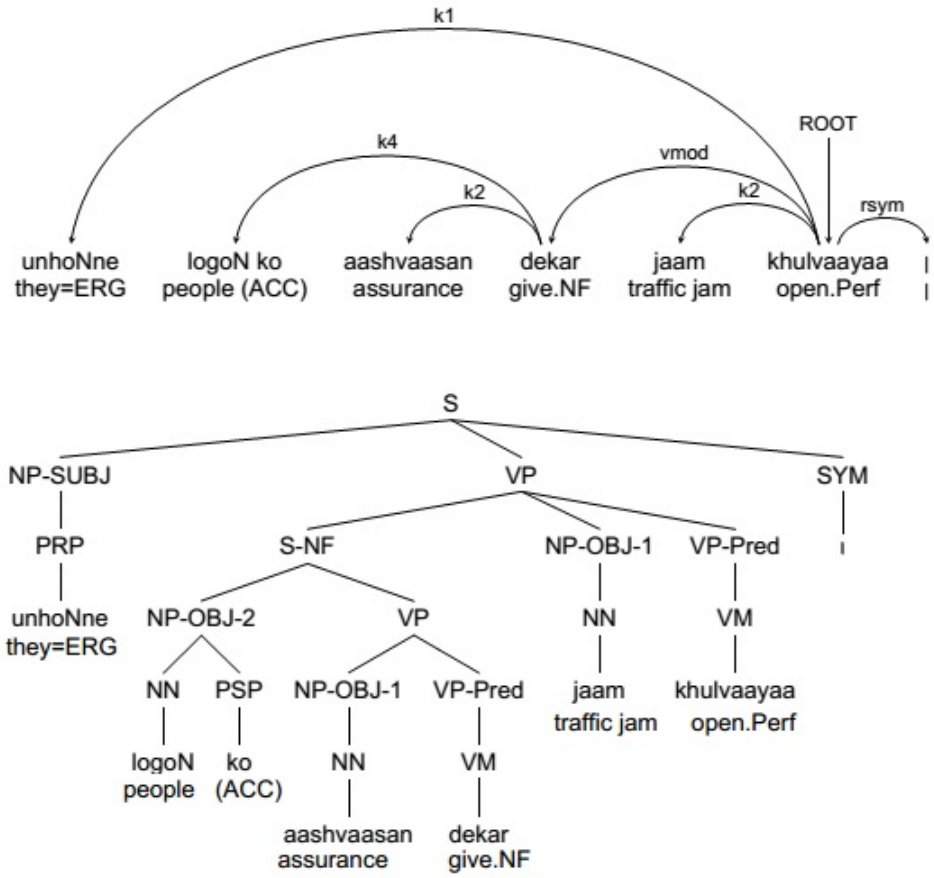

Figure 4: A DS and the derived PS showing a non-finite clause for the sentence 'After assuring the people they cleared up the traffic.'

3.3 Relative Clause

Similar to the DS representation, relative clauses directly modify the nouns in the PS tree (see Figure 5). We use the modifier-modified relation information from DS to identify the head of the relation clause. The S node in the relative clause tree appears as a sister of the head noun in the PS tree.

3.4 Multi-clause Coordination

Two clauses at the same level are adjoined to the coordinating conjunction such that both the clauses take the coordinating conjunction as their head (Figure 6).

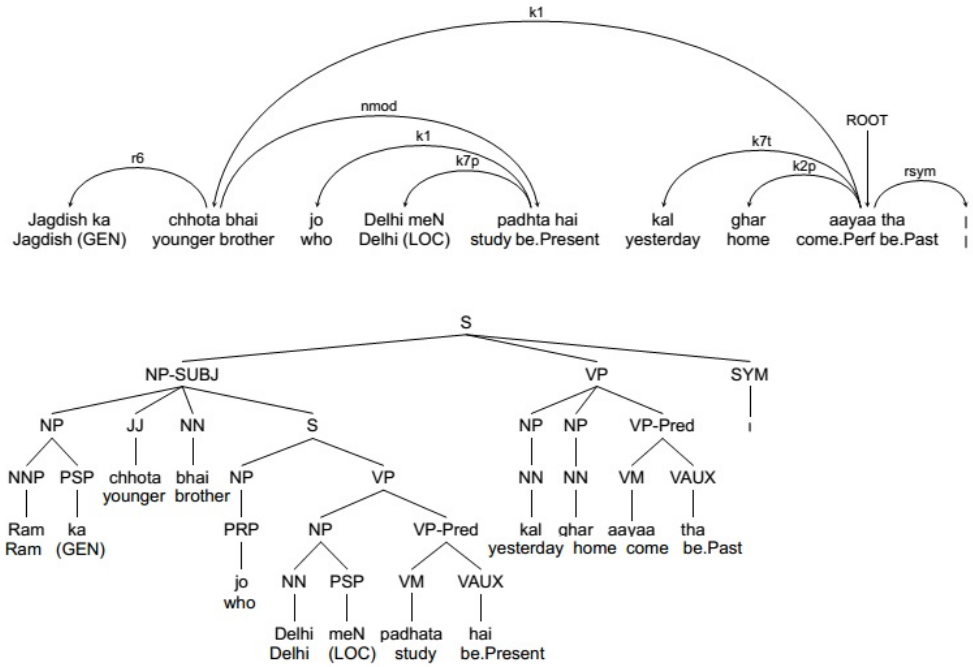

Figure 5: A DS and the derived PS showing an embedded relative clause for the sentence 'Jagdish's younger brother who studies in Delhi came yesterday'

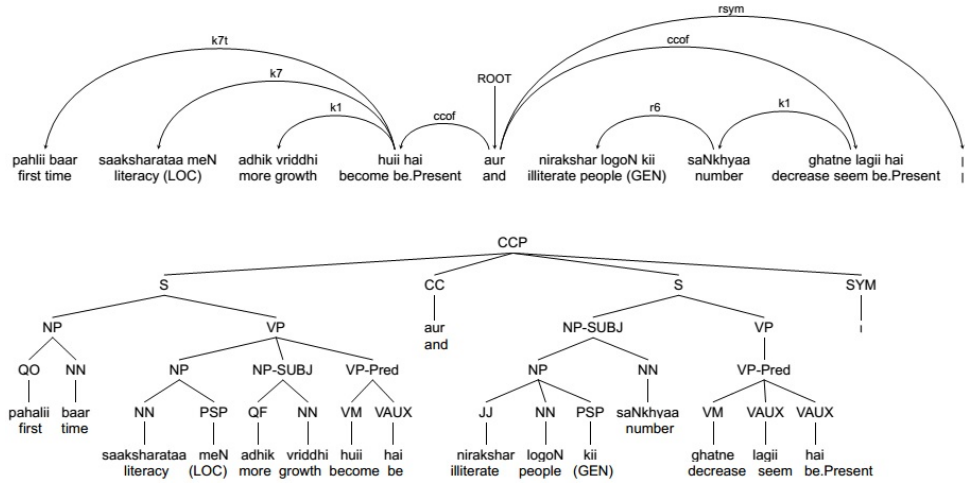

Figure 6: A DS and the derived PS showing a multi-clause coordination for the sentence 'For the first time literacy has increased significantly and the number of illiterates is going down.'

4 Evaluation

We evaluate the validity of the automatically generated PS trees using various metrics. This was done to ensure that the information transfer from DS to PS was indeed successful. In addition we wanted to test the structural consistency of the PS representation. This is necessary because we do not have gold PS trees, therefore we have to use quantitative methods to analyze the performance of our system. The evaluation was performed on Hindi Dependency Treebank (ver-0.05) which consists of 18853 sentences with an average of 20 words per sentence.[3]

We used the following metrics (1) Well-formedness: the tree has one root, is acyclic and has no duplicated sub-trees (2) Linear order: terminal nodes in the tree match the linear word order in sentence (3) Argument representation: All the arguments given in DS are reflected in the PS output (4) Clausal Correspondence: whether all verbal predicates in DS have their corresponding S nodes in PS.

The first two tests validate the correctness of the constituent structure while the latter two tests analyze the performance of our algorithm in extracting information from the DS and representing the same information on the PS.

Well-formedness	**Linear order**	**Argument representation**	**Clausal correspondence**	**All constraints**
99.90%	99.78%	99.87%	99.96%	99.74%

Table 2: System evaluation

Table 2 shows the performance of our conversion algorithm on all the metrics. We find that we are able to convert the DS trees into PS trees with high accuracy. The high accuracy demonstrates that the conversion produces PS trees that can be reliable enough for other applications. We note that our system is able to successfully convert nearly all types of constructions in the HUTB into PS. We also experimented with handling non-projectivity. We were successful in handling a majority of such dependencies. However the resulting PS structures were not always consistent with regards to the placement of the empty category. In addition, certain cases of inter-clausal non-projective dependencies were not being converted. In order to handle these issues consistently across the treebank, we chose to not handle non-projectivity in the current release of the implemented algorithm. We do plan to release an updated version that can handle non-projectivity in the near future.

5 Summary and Conclusion

In this paper, we have proposed a DS to PS conversion algorithm for the HUTB. The goal of our system was to maintain a close correspondence between the syntactic information in DS and PS representations. This algorithm generates PS

[3]We found 0.2% dependency trees in the treebank as either disjoint or cyclic which we ignored for the purposes of this evaluation. Non-projective trees were also not considered.

trees with high accuracy while avoiding an intermediate level of representation. We envisage that this resource will be immediately useful for NLP tools such as parsers and also for language modeling. Our system currently does not handle non-projectivity. In addition, we do not specifically handle complex predicates. We intend to handle these phenomena in the near future.

References

[1] Bharat R Ambati, Tejaswini Deoskar, and Mark Steedman. Hindi CCGbank: CCG treebank from the hindi dependency treebank. In *Language Resources and Evaluation*, 2016.

[2] Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, and Fei Xia. A multi-representational and multi-layered treebank for hindi/urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics, 2009.

[3] Rajesh Bhatt, Owen Rambow, and Fei Xia. Linguistic phenomena, analyses, and representations: Understanding conversion between treebanks. In *IJCNLP*, pages 1234–1242, 2011.

[4] Rajesh Bhatt and Fei Xia. Challenges in converting between treebanks: a case study from the hutb. In *META-RESEARCH Workshop on Advanced Treebanking*, page 53, 2012.

[5] Michael Collins, Lance Ramshaw, Jan Hajič, and Christoph Tillmann. A statistical parser for czech. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 505–512. Association for Computational Linguistics, 1999.

[6] Felix Engelmann, Lena A. Jäger, and Shravan Vasishth. The determinants of retrieval interference in dependency resolution: Review and computational modeling. Manuscript submitted, 2016.

[7] John T Hale. What a rational parser would do. *Cognitive Science*, 35(3):399–443, 2011.

[8] Philip Hofmeister and Ivan A Sag. Cognitive constraints and island effects. *Language*, 86(2):366, 2010.

[9] Alex Luu, Sophia A Malamud, and Nianwen Xue. Conversion of syntagrus dependency treebank into penn treebank style. In *Proceedings of the Tenth Linguistic Annotation Workshop*, pages 16–21. Association for Computational Linguistics, 2016.

[10] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

[11] C. Phillips, M. W. Wagers, and E. F. Lau. Grammatical illusions and selective fallibility in real-time language comprehension. In *J. Runner (ed.), Experiments at the Interfaces, Syntax and Semantics*, volume 37, pages 153–186. 2011.

[12] Owen Rambow. The simple truth about dependency and phrase structure representations: An opinion piece. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 337–340. Association for Computational Linguistics, 2010.

[13] Ashwini Vaidya, Jinho D Choi, Martha Palmer, and Bhuvana Narasimhan. Analysis of the hindi proposition bank using dependency structure. In *Proceedings of the 5th Linguistic Annotation Workshop*, pages 21–29. Association for Computational Linguistics, 2011.

[14] Fei Xia and Martha Palmer. Converting dependency structures to phrase structures. In *Proceedings of the first international conference on Human language technology research*, pages 1–5. Association for Computational Linguistics, 2001.