Creating a Tree Adjoining Grammar from a Multilayer Treebank

Rajesh Bhatt
Univ. of Massachusetts
Amherst, MA 01003, USA
bhatt@linguist.umass.edu

Owen Rambow
CCLS, Columbia University
New York, NY 10115, USA
rambow@ccls.columbia.edu

Fei Xia
Univ. of Washington
Seattle, WA 98195, USA
fxia@uw.edu

Abstract

We propose a method for the extraction of a Tree Adjoining Grammar (TAG) from a dependency treebank which has some representative examples annotated with phrase structures. We show that the resulting TAG along with corresponding dependency structure can be used to convert a dependency treebank to a TAG-based phrase structure treebank.

1 Introduction

In this paper, we address the problem of extracting a Tree Adjoining Grammar (TAG) from a treebank which has been annotated manually for dependency structure (DS), and which has a small set of representative example sentences manually annotated for both DS and phrase structure (PS). There has been much work on TAG extraction from PS treebanks (e.g., (Xia, 1999; Chen, 1999)). In those studies, heuristics are used to "cut up" the PS trees of entire sentences into elementary trees, which are then collected into a grammar.

This paper extends the previous work and makes the following novel contributions:

- We present a method for extracting a TAG from a DS treebank augmented with a set of representative examples of (DS, PS) pairs.
- We show that the resulting TAG paired with DS subtrees can be interpreted as conversion rules for converting the DS treebank to a PS treebank.

The structure of the paper is as follows. In Section 2, we review the relevant properties of the Hindi Treebank, which serves as the source of our examples in this paper. We then discuss the crucial notion of "consistency" in Section 3, and present our basic algorithm in Section 4. We then discuss two issues which require extensions to our agorithm: empty categories (Section 5) and long-distance word order variations (Section 6). These extensions are sketched, but we do not present them in full detail due to lack of space. Finally, we discuss remaining issues in Section 7 and then conclude.

2 The Hindi Treebank

In this paper, we use the Hindi Treebank (HTB) (Palmer et al., 2009) as an example, but the principles we present are language-independent. Compared to other existing treebanks, the HTB is unusual in that it contains three layers: dependency structure (DS), PropBank-style annotation (PB) (Kingsbury et al., 2002) for predicate-argument structure, and an independently motivated phrase-structure (PS) annotation which is automatically derived from the DS plus the PB. The treebank is created in three steps. The first step is the manual annotation of DS. The second step is PropBanking, which focuses on adding the lexical predicate-argument structure on top of DS. The third step is the automatic creation of PS, which is done by a DS-to-PS conversion process that takes DS and PB as input and generates PS as output.

Figure 1 (top row) shows the three layers for Ex (1), a sentence with the unaccusative verb *break*. Because the three layers are independently motivated, they have a certain freedom in choosing how they represent syntactic phenomena. In this example, the DS layer decides to treat unaccusative verbs the same way as unergative (such as *sleep*), and marks their subjects as a *k1*. In contrast, the PB makes the distinction, marking the subject of an unaccusative verb as an *ARG1*, not *ARG0* (as is the case for the subjects of unergative verbs); PS also makes the distinction, by in-

Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11), pages 162–170,
Paris, September 2012.

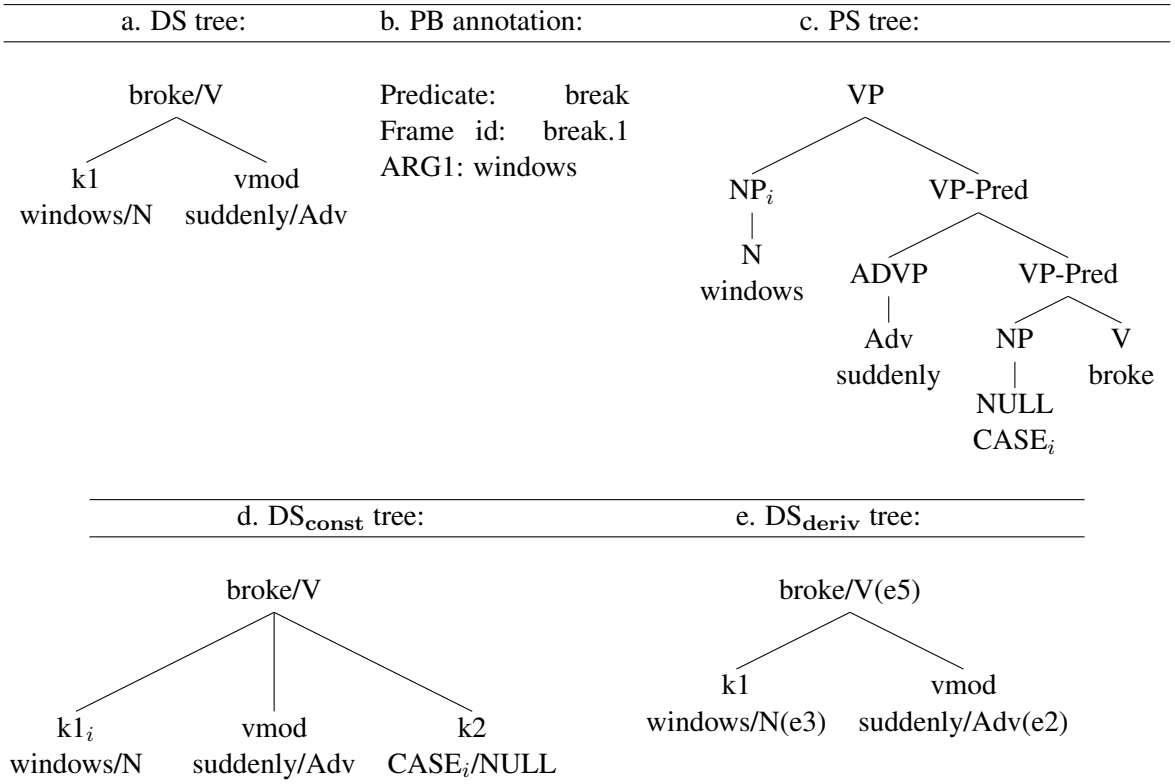

Figure 1: The three annotation levels of the HTB: Dependency Structure, PropBank, and Phase Structure (top row); the consistent DS (DS$_{\mathbf{const}}$) and the derivation tree (DS$_{\mathbf{deriv}}$) obtained from these structures (bottom row); e_i in the DS$_{\mathbf{deriv}}$ tree refers to elementary trees in Figures 5-6.

dicating *movement* from the object position to the subject position using a coindexed trace.

(1) khiṛkiyã: acaanak ṭuuṭĩ:
windows.F suddenly broke.FPl
'The windows broke suddenly.'

3 Consistency between DS and PS

In our previous study (Xia et al., 2009; Bhatt et al., 2011), we proposed a DS-to-PS conversion algorithm, which extracts conversion rules from a small number of (DS, PS) pairs, and then applies the rules to a new DS to generate a PS for the DS.

The algorithm introduces two concepts: *consistency* and *compatibility*. A DS and a PS tree are *consistent* if and only if there exists an assignment of head words for the internal nodes in the PS such that after merging all the (head child, parent) nodes in the PS, the new PS is identical to the DS. In Figure 1, the DS in (a) and the PS in (c) are not consistent because the empty category *CASE* appears only in (c). In contrast, the dependency structure in (d), where *CASE* is added to the original DS as a *k2* dependent of the verb, is consistent with the PS in (c). A DS and a PS analysis for a linguistic phenomenon are *compatible* if the (DS,PS) tree pairs for the sentences with that phenomenon are consistent.

Bhatt and Xia (2012) demonstrate that automatic, high-quality DS-to-PS conversion is facilitated when the analyses chosen by the DS and PS guidelines for most linguistic phenomena are compatible; for the phenomena with incompatible analyses, manually written rules transform DS to an intermediate representation called DS$_{\mathbf{const}}$ (*const* stands for *consistency*). The process of creating DS$_{\mathbf{const}}$ from a DS is explained in that paper.

We are currently using the algorithm to automatically generate the PS trees in the HTB, given the manual annotation of the DS and PB layers. The process is illustrated in Figure 2.[1] The process assumes that there is a small set of sentences with all three layers of annotation, which are used in the training stage of the conversion to extract

[1] PB is part of the input for the conversion process because it provides certain information (e.g., unaccusative vs. unergative distinction) that is not present in the DS. For the sake of simplicity, we still call the process "DS-to-PS conversion", though "DS+PB-to-PS conversion" would be more complete.

conversion rules. The rules are then applied to the DS and PB for the new sentences in the test stage to generate PS. For instance, if the DS and PB in Figure 1a. and 1b. are the input to the training stage, the system will create $\text{DS}_{\mathbf{const}}$ in Figure 1d. From the ($\text{DS}_{\mathbf{const}}$, PS) pair (Figure 1d. and 1c.), the system will automatically extract the conversion rules in Figure 3. If these rules are applied to the same $\text{DS}_{\mathbf{const}}$ tree in the test stage, the PS created by the system will be identical to the one in Figure 1c. The details of the algorithm can be found in (Xia et al., 2009; Bhatt and Xia, 2012).

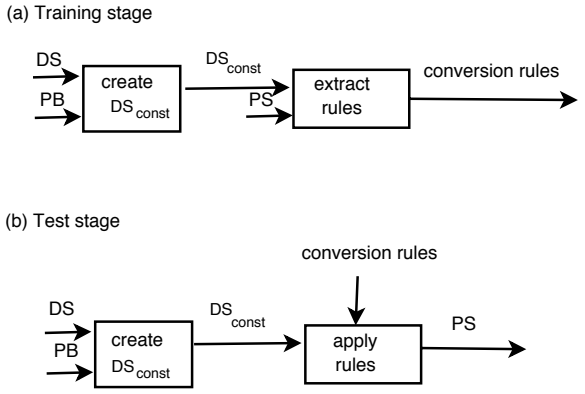

Figure 2: The flow chart for DS-to-PS conversion

4 Basic TAG Extraction Algorithm

Given a dependency treebank *DTB* and a set of (DS, PS) pairs, the goals of this study are (1) to design the algorithm that extracts a TAG from the DTB, and (2) transforms the DTB into a treebank for the extracted TAG–that is, for each sentence in the DTB, we want to create a derivation tree and a derived tree for the sentence based on the extracted TAG with the requirement that the derived tree is consistent with the $\text{DS}_{\mathbf{const}}$ for the sentence.

One possible approach works as follows: (1) run the DS-to-PS conversion algorithm described in Section 3 on the DTB to generate a PS treebank; (2) extract a TAG from the PS treebank by adapting an existing grammar extraction algorithm (e.g., (Xia, 1999; Chen, 1999)); (3) run a TAG parser to generate a set of parse trees for each sentence in the DTB; (4) for each sentence in the DTB, to maintain the dependency relation in its DS, a filter is needed to throw away all the parse trees generated in (3) that are not consistent with the $\text{DS}_{\mathbf{const}}$ for that sentence.

This approach has several limitations. First, it requires adapting a grammar extraction algorithm in Step (2) and writing a filter in Step (4). Second, it is highly inefficient; for instance, many parse trees generated in Step (3) will later be thrown away in Step (4). Third, the connection between dependency relation in the DTB and elementary trees in the extracted TAG is not represented as the latter is extracted from the PS produced in Step (1), not from the DS directly.

We propose a new approach (see Figure 4), which not only extracts a TAG from a dependency treebank directly, but also provides a new way for converting DS to PS. Compared to the DS-to-PS conversion algorithm (see Figure 2), the second modules in the training and test stages are different and the definition of conversion rules is extended. In the original definition, the lefthand side of a conversion rule is a dependency link, and the righthand side is similar to a context-free rule. In the new definition, the lefthand side is a subtree of a DS that includes a head and either all its arguments or one adjunct with the adjunct's arguments, and the righthand side is an elementary tree. We call the new rule *etree-based rule* (*etree* for elementary tree).

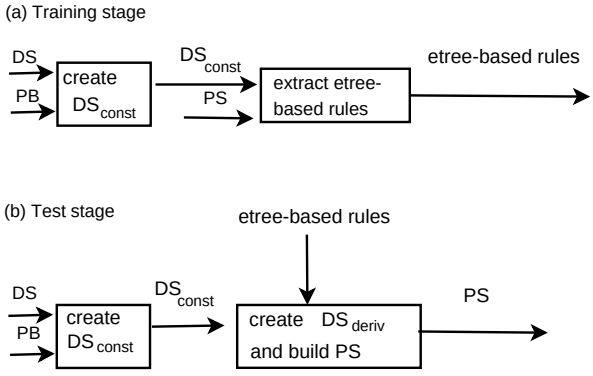

Figure 4: The flow chart for converting the HTB into a TAG-based treebank

4.1 Extracting *etree-based* conversion rules and TAGs

The training stage, as in Figure 4(a), has two modules. The first module is the same as the first module in the DS-to-PS conversion algorithm and is described in (Bhatt and Xia, 2012). The second module takes ($\text{DS}_{\mathbf{const}}$,PS) pairs as input and outputs etree-based rules, and Table 1 shows its pseudocode.

Xia et al. (1999) described an algorithm that extracts TAGs from a PS treebank. Given a phrase structure T, the algorithm extracts a TAG in four

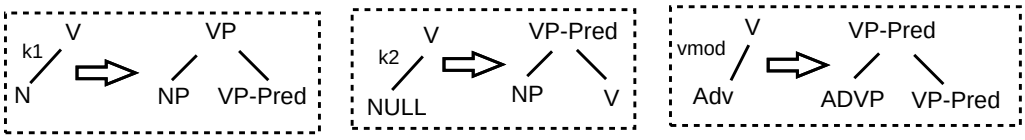

Figure 3: Conversion rules extracted from the ($DS_{\mathbf{const}}$,PS) pair in Figure 1, which is produced by the DS-to-PS conversion process in Figure 2.

Input: *TreePairs* is a set of ($DS_{\mathbf{const}}$, PS) pairs
ArgTable is a set of (head-POS-tag, dep-types)

Output: *ConvRules* is a set of *etree-based* conversion rules

Algorithm: ExtConvRules(TreePairs, ArgTable)
(1) *ConvRules* = ϕ;
(2) for each ($DS_{\mathbf{const}}$, PS) pair in *TreePairs*
(3) // Step 1: Make argument/adjunct distinction in the $DS_{\mathbf{const}}$
(4) for each internal node h in $DS_{\mathbf{const}}$
(5) mark each child d of h as argument or adjunct based on *ArgTable*

(6) // Step 2: Choose head child and make argument/adjunct distinction in the PS
(7) for each dependency link (d, h) in $DS_{\mathbf{const}}$, do the following in the PS
(8) find the lowest ancestor of d and h in PS and call it Y
(9) let X_d be Y's child that dominates d and X_h be Y's child that dominates h
(10) mark X_h as the head child of Y
(11) mark X_d as an argument or adjunct child of Y based on the label of d in line (5)

(12) // Step 3: deflat the PS and extract *etrees* from it
(13) *deflat* the PS so that arguments and adjuncts are not siblings
(14) extract *etrees* from the deflatted PS

(15) // Step 4: create conversion rules
(16) for each node h in $DS_{\mathbf{const}}$
(17) for each adjunct child d of h
(18) create a rule where the lefthand side is a subtree of $DS_{\mathbf{const}}$ with dependency
(19) link (d, h) and links between d and all its argument children (if any),
(20) and the righthand side is the corresponding auxiliary tree anchored by d
(21) if (h is the root of $DS_{\mathbf{const}}$) or (h is not marked as an adjunct child of its parent)
(22) create a rule where the lefthand side is h with all its argument children in the
(23) $DS_{\mathbf{const}}$, and the righthand side is the corresponding initial tree anchored by h
(24) add the rules to *ConvRules*

(25) return *ConvRules*

Table 1: Algorithm for extracting *etree-based* conversion rules

Input: *ConvRules* which is extracted from the training stage
ArgTable is the same as the one used in the training stage
A new $\mathrm{DS}_{\mathbf{const}}$

Output: A set of PSs for the input $\mathrm{DS}_{\mathbf{const}}$

Algorithm: BuildPS(ConvRules, ArgTable, $\mathrm{DS}_{\mathbf{const}}$)
(1) $\mathrm{DS}_{\mathbf{deriv}}$= $\mathrm{DS}_{\mathbf{const}}$
(2) $\mathrm{DS}_{\mathbf{deriv}}$ has an extra field EtreeSet[n] that stores the possible etrees for node n

(3) // Step 1: make argument/adjunct distinction in $\mathrm{DS}_{\mathbf{const}}$
(4) for each internal node h in $\mathrm{DS}_{\mathbf{deriv}}$
(5) mark each child d of h as argument or adjunct based on *ArgTable*

(6) // Step 2: Find *etrees* for the nodes in $\mathrm{DS}_{\mathbf{const}}$
(7) for each node h in $\mathrm{DS}_{\mathbf{const}}$
(8) for each adjunct child d of h
(9) find rules in *ConvRules* whose lefthand side is a subtree of $\mathrm{DS}_{\mathbf{const}}$ that
(10) includes the dependency link (d, h) and links between d and its arguments
(11) EtreeSet[d] = the etrees in these matched rules
(12) if (h is the root of $\mathrm{DS}_{\mathbf{const}}$) or ($h$ is not marked as an adjunct child of its parent)
(13) find rules in *ConvRules* whose lefthand side is a subtree of $\mathrm{DS}_{\mathbf{const}}$ that
(14) includes h and all its argument children (if any)
(15) EtreeSet[h] = the etrees in these matched rules

(16) // Step 3: Build PSs from $\mathrm{DS}_{\mathbf{deriv}}$
(17) PSset = ϕ
(18) for each combination of etrees in $\mathrm{DS}_{\mathbf{deriv}}$
(19) (In a *combination*, one etree is picked from EtreeSet[n] for each node n in $\mathrm{DS}_{\mathbf{deriv}}$)
(20) generate PSs from the combination
(21) add these PSs to PSset

(22) return PSset

Table 2: Algorithm for building PSs for a given $\mathrm{DS}_{\mathbf{const}}$

steps: (i) for each internal node n in T, select one of its children as head child according to a head percolation table; (ii) for other children of n, determine whether each child is an argument or an adjunct of n using an argument table; (iii) *deflat* the PS so that arguments and adjuncts are not siblings; (iv) extract etrees from deflatted PS by decomposing the PS into pieces and gluing some pieces together to form etrees.

The algorithm in Table 1 can be seen as an extension of the algorithm in (Xia, 1999). First, because its input includes $\mathrm{DS}_{\mathbf{const}}$, the dependency relation is already given between words so Step (i) is no longer needed; second, argument/adjunct distinction is made on $\mathrm{DS}_{\mathbf{const}}$ first (see Step 1) and then carried over to PS (see Step 2). Step 1 uses an argument table which specifies what kind of dependents are considered as arguments of a head. For instance, if dependents with *k1*, *k2*, or *k4* role labels are considered arguments of a verb, the argument table will include an entry *(v, k1/k2/k4)*. Step 2 makes the argument/adjunct distinction in the PS based on the decision made in the DS. Step 3 is identical to Steps (iii)-(iv) and it extracts etrees from the deflatted PS. Step 4 links subtrees in the $\mathrm{DS}_{\mathbf{const}}$ to extracted etrees to form etree-based rules. Each subtree in an etree-based rule includes a head and either all its arguments (this determines how many substitution nodes the etree has) or one adjunct of the head and the adjunct's arguments (if any). The righthand side of the etree-based rules form a TAG.

Given the ($\mathrm{DS}_{\mathbf{const}}$, PS) pair in Figure 1d and 1c, the extracted rules are shown in Figure 5. Compared to the rules in Figure 3, etree-based rules have extended locality.

4.2 Generating PS from DS

The test stage (see Figure 4(b)) also has two modules: the first one is the same as the one in the training stage. Table 2 shows the algorithm for the second module. The algorithm has three steps. The first step makes argument/adjunct distinction in the input $\mathrm{DS}_{\mathbf{const}}$.

The second step finds the etrees that each word in the $\mathrm{DS}_{\mathbf{const}}$ could anchor. This is done by forming a subtree of $\mathrm{DS}_{\mathbf{const}}$ that a word belongs to (similar to Step 4 in Table 1) and then selecting conversion rules that match that subtree. The etrees in those selected conversion rules are stored with the node in $\mathrm{DS}_{\mathbf{const}}$, resulting in a new tree called $\mathrm{DS}_{\mathbf{deriv}}$. The subscript *deriv* in $\mathrm{DS}_{\mathbf{deriv}}$ stands for *derivation tree* because $\mathrm{DS}_{\mathbf{deriv}}$ is just like the derivation tree in the TAG except that it does not include the positions of substitution/adjoining operations and which etree the word anchors in the current sentence is not determined.

The third step produces a set of PSs from the $\mathrm{DS}_{\mathbf{deriv}}$. The set could have more than one PS due to various types of ambiguity. In principle, one could imagine that there is already a one-to-many mapping between the $\mathrm{DS}_{\mathbf{const}}$ the PS. We do not handle ambiguities that have such a source as we consider these to be a treebank design flaw. Either side should be enriched or modified during the design process to eliminate intrinsically ambiguous mappings. However, ambiguity can arise despite careful design of the levels of representation. For example, in Hindi adverbs can occur before or after the subject. From the DS and PS trees for a sentence such as (1), the rule with adjunction at the VP-Pred-level will be extracted (the second rule in Figure 5), while the DS and PS trees for *acaanak khiṛkiyā: ṭuuṭī:* ("suddenly the windows broke") yields a rule with adjunction at the VP-level. Thus, we have an ambiguous rule. Most of the time this does not matter since word order resolves the adjunction level, but if the subject is empty we obtain two possible derivations using two different etrees. Another, related type of ambiguity arises when there are multiple instances of a node label in one tree, allowing for multiple derivation using the *same* etrees. Consider a simple case of a verb that has a left adjunct as well as a right adjunct: **L V R**. Let us assume that, according to the conversion rules, both **L** and **R** can adjoin at the VP level. This will lead to two PSs, one with **L** attached higher than **R**, and the other with **R** attached higher than **L**.

5 Empty Categories

In the basic algorithm outlined in Section 4, $\mathrm{DS}_{\mathbf{deriv}}$ is isomorphic to $\mathrm{DS}_{\mathbf{const}}$ by design. The problem with that design is that $\mathrm{DS}_{\mathbf{const}}$ may include nodes labeled with empty categories (ECs) (see Figure 1d.), meaning that some etrees in the resulting TAG would be "lexicalized" by an EC and that could cause difficulty when using the TAG for parsing.

To solve this problem, we make two minor revisions to the basic algorithm: in the rule extraction

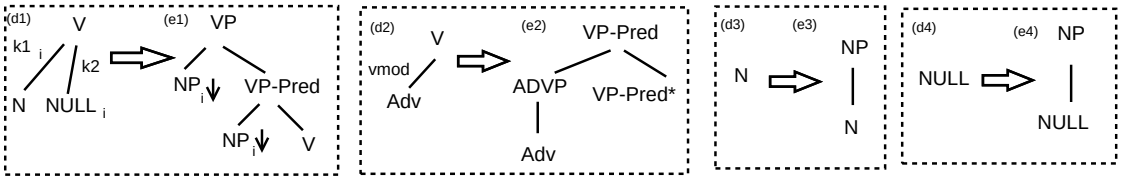

Figure 5: Etree-based rules extracted by the algorithm in Table 1 from the ($DS_{\mathbf{const}}$,PS) pair in Ex (1)

module (see Table 1), if in the $DS_{\mathbf{const}}$ a head has a dependent which is an EC, then the EC (along with its phrase structure projection, if any) is included in the etree for the head word (see the etree in Figure 6). Similarly, in the creating $DS_{\mathbf{deriv}}$ module (see Table 2), the EC leaf node will be removed from $DS_{\mathbf{deriv}}$ as it is already included in the etree for its parent node.

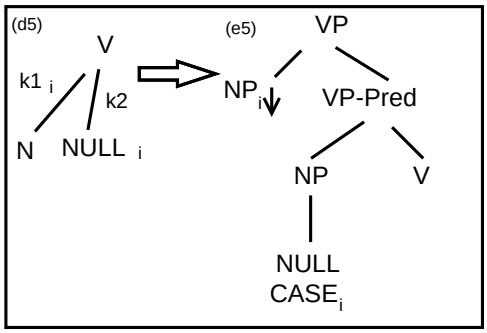

Figure 6: The new etree-based rule extracted from the ($DS_{\mathbf{const}}$, PS) pair in Figure 1. It will replace the first and last rules in Figure 5

In the case of our unaccusative example, the extracted TAG will include the rule in Figure 6, which replace the first and last rules in Figure 5. The new $DS_{\mathbf{deriv}}$ is in Figure 1e., where e_2 and e_3 are two etrees in Figure 5, and e_5 is the etree in Figure 6.[2]

6 Issues in Word Order

It has been known for a long time that in a lexicalized TAG, the derivation tree is a lexical dependency tree (since the nodes are bijectively identified with the words of the sentence), and that this dependency structure is not necessarily the linguistically plausible structure (Rambow and Joshi, 1997; Candito and Kahane, 1998; Rambow et al., 2001). Consider the following example (2):

(2) kitaabẽ mẼ-ne khariid-nii caah-ii
books.F I-Erg buy-Inf.F want-Pfv.F
'Books, I had wanted to buy.'

Here, the standard analysis is that the matrix clause anchored on *want* is adjoined into the embedded clause anchored on *buy*, which results in a derivation structure in which the *buy* node dominates the *want* node, contrary to the dependency representation. Furthermore, it has been known that if we want the etrees and the derivation structure to have a linguistic interpretation, then certain word orders in free word order languages cannot be derived using a TAG (Rambow, 1994; Chen-Main and Joshi, 2008). Consider the following example.

(3) mẼ-ne yeh kitaab sab-se khariid-ne-ko
I-Erg this book all-Instr buy-Inf-Acc
kah-aa
say-Pfv
'I told everyone to buy this book.'

In (3), *this book* is an argument of the embedded verb *buy*, and is placed between the matrix verb's two arguments. While this can be derived by a TAG, it would require an etree headed by *buy* to have a VP foot node, which has no plausible linguistic interpretation (since *buy* does not have a clausal argument).

Both problems have a common solution: non-local multicomponent TAGs with dominance links (nlMC-TAG-DL), for example in the definition given in (Rambow et al., 2001) as D-Tree Substitution Grammars (DSG). In a DSG, there is only the substitution operation, but all links in trees are interpreted as non-immediate dominance links; in fact, the trees can be seen as tree descriptions rather than as trees; this allows components of trees to move up and be inserted into other trees. For large-scale grammar development in DSG, see (Carroll et al., 1999).

There are good reasons to want to try and restrict the generative power of a formal system used

[2]It is not a coincident that in this example $DS_{\mathbf{deriv}}$ and DS are isomorphic, as the DS-to-$DS_{\mathbf{const}}$ module involves adding ECs and $DS_{\mathbf{const}}$-to-$DS_{\mathbf{deriv}}$ involves removing ECs. But in theory, $DS_{\mathbf{deriv}}$ and DS are not necessarily always isomorphic as the DS-to-$DS_{\mathbf{const}}$ module is not limited to adding ECs.

for the description of syntax: on the one hand, we may want to restrict the formal power in order to obtain parsing algorithms of certain restricted complexities (Kallmeyer, 2005), and on the other hand, we may want to make assumptions about the underlying formalism in order to make predictions about what word orders are grammatical (or plausible) (Chen-Main and Joshi, 2008). However, in this paper we do not address the tradeoff between non-local and restricted MC-TAG systems, and we do not present empirical arguments from Hindi in order to address the issue of what formal complexity is required for the syntactic description of Hindi. We leave those issues to future work. Instead, we assume a simple framework in which we can explore the issue of grammar extraction.

The algorithm for extraction of a TAG can be extended straightforwardly to an algorithm for extracting a DSG: whenever in $\mathrm{DS}_{\mathbf{const}}$ we have a node labeled with an empty category which is coindexed with a node which is higher in the tree (in a c-command relationship), then we put the two etrees resulting from those two nodes into one set, and add a dominance relation. The step for creating $\mathrm{DS}_{\mathbf{deriv}}$ should also be revised accordingly to indicate that the two etrees belong to one set.

7 Conclusion

We have given a basic algorithm for the extraction of a TAG from a dependency treebank which has some representative examples annotated with phrase structures. This TAG can be used for any purpose TAGs can be used, such as parsing or generation, but we can also use etree-based conversion rules to convert the entire DS treebank to a TAG-based treebank. We have also sketched how this basic algorithm can be extended to handle empty categories and word order variation.

There are several important remaining issues. The first issue is ambiguity. Ideally, we want to have exactly one PS and one TAG derivation tree for each $\mathrm{DS}_{\mathbf{const}}$ in the test stage because we can then transform the original dependency treebank into a treebank for the extracted TAG. However, as discussed in Section 4.2, the second module in the test stage may produce multiple PSs for a given $\mathrm{DS}_{\mathbf{const}}$. One possible way to reduce spurious ambiguity is to look at the PS trees in the training data where ambiguity of attachment can arise. If there are regularities in these situations (for example, lowest attachment possible, and a certain order for attaching left and right adjuncts), then that information could be passed to the test stage to reduce ambiguity.

The second issue is unseen rules. The etree-based rules are extracted from the ($\mathrm{DS}_{\mathbf{const}}$, PS) pairs in the training stage. If the number of such representative examples is small, it is unlikely that the rules extracted from the pairs are complete enough to cover the $\mathrm{DS}_{\mathbf{const}}$ subtrees in the test stage. Some kind of backoff strategy is needed to handle such cases. We will study both issues in the future.

Acknowledgment This work is supported by NSF grants CNS-0751089, CNS-0751171, and CNS-0751213. We would like to thank the anonymous reviewers for helpful comments and our colleagues on the Hindi-Urdu Treebank Project for their support.

References

Rajesh Bhatt and Fei Xia. 2012. Challenges in converting between treebanks: a case study from the hutb. In *Proceedings of META-RESEARCH Workshop on Advanced Treebanking, in conjunction with LREC-2012*, Istanbul, Turkey.

Rajesh Bhatt, Owen Rambow, and Fei Xia. 2011. Linguistic phenomena, analyses, and representations: Understanding conversion between treebanks. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP)*, pages 1234–1242, Chiang Mai, Thailand.

Marie-Hélène Candito and Sylvain Kahane. 1998. Can the TAG derivation tree represent a semantic graph? An answer in the light of Meaning-Text Theory. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, IRCS Report 98–12, pages 21–24. Institute for Research in Cognitive Science, University of Pennsylvania.

John Carroll, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir. 1999. Parsing with an extended domain of locality. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 217–224.

Joan Chen-Main and Aaravind K. Joshi. 2008. Flexible composition, multiple adjoining and word order variation. In *Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 9), Tübingen, Germany*.

John Chen. 1999. An Investigation into Efficient Statistical Parsing Using Lexicalized Grammatical Information. Thesis proposal, University of Delaware.

Laura Kallmeyer. 2005. Tree-local multicomponent tree-adjoining grammars with shared nodes. *Comput. Linguist.*, 31(2):187–226, June.

Paul Kingsbury, Martha Palmer, and Mitch Marcus. 2002. Adding semantic annotation to the Penn TreeBank. In *Proceedings of the Human Language Technology Conference (HLT-2002)*, San Diego, CA.

Martha Palmer, Rajesh Bhatt, Bhuvana Narasimhan, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. Hindi Syntax: Annotating Dependency, Lexical Predicate-Argument Structure, and Phrase Structure. In *Proceedings of ICON-2009: 7th International Conference on Natural Language Processing*, Hyderabad.

Owen Rambow and Aravind Joshi. 1997. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Leo Wanner, editor, *Recent Trends in Meaning-Text Theory*, pages 167–190. John Benjamins, Amsterdam and Philadelphia.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Tree Substitution Grammars. *Computational Linguistics*, 27(1).

Owen Rambow. 1994. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Available as Technical Report 94-08 from the Institute for Research in Cognitive Science (IRCS) and also at ftp://ftp.cis.upenn.edu/pub/rambow/thesis.ps.Z .

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2009. Towards a multi-representational treebank. In *The 7th International Workshop on Treebanks and Linguistic Theories (TLT-7)*, Groningen, Netherlands.

Fei Xia. 1999. Extracting Tree Adjoining Grammars from Bracketed Corpora. In *Proc. of 5th Natural Language Processing Pacific Rim Symposium (NLPRS-1999)*, Beijing, China.