Challenges in Converting between Treebanks: a Case Study from the HUTB

Rajesh Bhatt* **and Fei Xia†**

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

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

Abstract

An important question for treebank development is whether high-quality conversion from one representation (e.g., dependency structure) to another representation (e.g., phrase structure) is possible, assuming that annotation guidelines exist for both representations. In this study, we demonstrate that the conversion is possible only under certain conditions, and even when the conditions are met, the conversion is complex as we need to examine the two sets of guidelines on a phenomenon-by-phenomenon basis and provide an intermediate representation for phenomena with incompatible analysis.

1. Introduction

There has been much interest in converting treebanks from one representation to another; for instance, from phrase structure to dependency structure or from phrase structure to other grammatical frameworks such as LTAG, HPSG, CCG, or LFG. While there have been many studies on converting between treebank representations (Collins et al., 1999; Xia and Palmer, 2001; Cahill et al., 2002; Nivre, 2003; Hockenmaier and Steedman, 2007), it is not clear how well the proposed conversion algorithms work because, for the treebanks used in those studies, annotation guidelines are available only for one of the two representations.

Compared to other existing treebanks, the Hindi/Urdu Treebank (HUTB) (Palmer et al., 2009) 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. Because the treebank has detailed guidelines for all three layers and hundreds of guideline sentences with all three layers manually annotated, the treebank is a good resource for evaluating the performance of conversion algorithms. More importantly, the DS guidelines and the PS guidelines are based on different linguistic theories and the DS and the PS, as two representations, have different properties. While the idea of automatically creating PS trees from the DS and PB is appealing as it reduces the amount of human annotation, it raises many interesting questions:

- Does a *general-purpose*, high-quality DS-to-PS conversion algorithm exist? That is, an algorithm that performs well for any given sets of DS and PS guidelines?
- How much "freedom" do the designers of the DS and PS guidelines have in choosing analyses for linguistic phenomena?
- What kind of information should be included in the DS and PB in order to make the automatic conversion possible?

These questions are difficult to answer in the abstract. In this paper, we discuss them in the context of our experiences with the construction of the multi-representational Hindi-Urdu Treebank which involves automatic generation of the PS from the DS and PB.

2. An Overview of the HUTB

The HUTB (Palmer et al., 2009) has been developed by our colleagues and us since 2008. It has three layers, as explained below.

2.1. Dependency Structure (DS)

The HUTB chose the Paninian grammatical model (Bharati et al., 1995; Begum et al., 2008) as the basis of the DS analysis. The sentence is treated as a series of modifier-modified relations which has a primary modified (generally the main verb). The relations are of two types: karaka (roles of various participants in an action, i.e., arguments, notated as k1-k6) and others (roles such as purpose and location, i.e. adjuncts).

2.2. Propbank (PB)

PropBanking is a semantic layer of annotation that adds predicate argument structures to syntactic representations (Palmer et al., 2005). For each verb, PropBank represents the information about the arguments that appear with the verb in its corresponding frame file. The arguments of the verbs are labeled using a small set of numbered arguments, e.g. Arg0, Arg1,

Arg2, etc. Additionally, verb modifiers are annotated using functional tags such as ArgM-LOC, ArgM-TMP, ArgM-MNR.

2.3. Phrase Structure (PS)

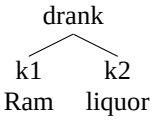
The PS guidelines are inspired by the Principles-and-Parameters methodology of Chomsky (1981). PS assumes a binary branching representation, where a minimal clause distinguishes at most two positions structurally (the core arguments). Displacement of core arguments from their canonical positions is represented via traces.

2.4. Overall Process

The treebank has three sets of annotation guidelines, one for each layer. 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 the 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 PropBank as input and generates PS as output. Figure 1 shows the three layers for a simple sentence. For the sake of saving space and readability for non-Hindi speakers, Hindi sentences in this paper are written as English words in Hindi word order.

(1) a. Ram liquor drank ('Ram drank liquor')

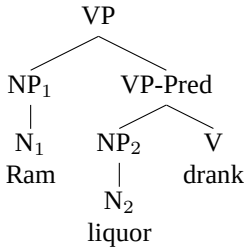
b. DS tree:



c. PB annotation:

Predicate: drank
Frame id: drink.1
Arg0: Ram
Arg1: liquor

d. PS tree:



3. DS-to-PS Conversion

While the input to the process includes DS and PB, for the sake of simplicity, in the rest of the paper we will simply call the process *DS-to-PS conversion*, with the understanding that the *DS* in this context also includes information from the PB.

3.1. Previous work on DS-to-PS conversion

The common setting of a DS-to-PS conversion process is given in Figure 1, which has three stages. In the training stage, the input is a set of (DS, PS) pairs, $\{(DS_i, PS_i)\}$; the output is a model, a set of conversion rules, or something else depending on the conversion algorithm. In the test stage, a DS tree, DS_t, is sent to the test module, along with the output of the training stage; the test module produces a PS tree, $PS_t^{(0)}$. In the evaluation stage, the output of the test stage is compared with the gold standard, PS_t, and some scores (e.g., labeled F-score) are produced as a measure for the overall performance of the conversion algorithm.

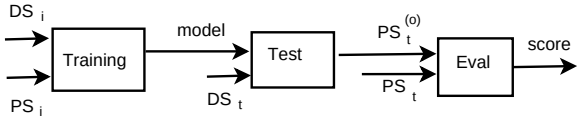


Figure 1: DS-to-PS conversion and evaluation

The previous DS-to-PS conversion algorithms can be divided into two types depending on whether there is an explicit training stage. In (Collins et al., 1999; Xia and Palmer, 2001), the conversion algorithms were purely rule-based: the rules were created by hand and used to build $PS_t^{(0)}$ given DS_t; there was no training stage. Xia et al. (2009) automated the conversion process by introducing the concept of *consistency* between a DS and a PS and proposing a process that extracts conversion rules from consistent (DS, PS) pairs in the training stage; in the test stage, the extracted rules were applied to an input DS_t to generate $PS_t^{(0)}$. One limitation of these previous studies is that they evaluated their conversion algorithms on treebanks for which annotation guidelines and manual annotation exist only for one of the two representations, and, therefore, it is not clear how well the algorithms truly performed.

Bhatt et al. (2011) proposed an analytical framework for determining how difficult it would be to convert one representation to another representation (DS and PS in this case) when each representation has its own annotation guidelines. They demonstrated that the conversion procedure must examine guidelines on a phenomenon-by-phenomenon basis, and for each phenomenon, there are three possible scenarios: (1) the two guidelines have *compatible* analyses; (2) they have incompatible analyses; and (3) one represents the phenomenon but the other does not. In the first case, automatic conversion is fairly direct; in the second case, one needs to study the DS and PS analyses for the phenomenon and provide an intermediate representation to bring the gap; in the third case, additional information is required to achieve the conversion.

Bhatt et al. (2011) defined *compatibility* of analyses based on *consistency* of (DS,PS) pairs. As defined in (Xia et al., 2009), a PS and a DS are called *consistent* if and only if there exists an assignment of head words for the internal nodes in PS such that after the flatten operation and the label replacement operation, the new PS is identical to the DS once we ignore the depen-

dency types in the DS.[1] For instance, the DS and the PS in Ex (1) are consistent because when we choose *V* as the head child of *VP-Pred*, and *VP-Pred* as the head child of *VP*, we will merge these three nodes in the flatten operation and relabel the merged node with the head word *drank*; similarly, N_1 and NP_1 are merged and relabeled as *Ram*, and N_2 and NP_2 are merged and relabeled as *liquor*. The resulting tree is identical to the DS tree if we ignore the dependency types.

Given a linguistic phenomenon, let *D* be the set of (DS, PS) pairs for the sentences in the guidelines for that phenomenon. The analyses in the DS and PS guidelines are called *compatible* if and only if every (DS, PS) pair in *D* is consistent.

3.2. Our conversion process

Before we get into the details of our conversion process, it is important to address the first question raised in Section 1.: does a general-purpose DS-to-PS conversion algorithm exist that works well for any given sets of DS and PS guidelines? Referring to the flowchart in Figure 1, a conversion algorithm would correspond to the training and test modules; the DS and PS guidelines would correspond to $(DS_{i/k}, PS_{i/k})$ pairs; a *general-purpose, high-quality* algorithm would be one that produces $PS_t^{(0)}$ that is very similar to PS_t and therefore leads to a high evaluation score, no matter what $(DS_{i/k}, PS_{i/k})$ pairs look like.

Note that the flowchart shows the same setting as any machine learning (ML) system if we just replace *DS* with the input of a ML task (e.g., *sentence* for the parsing task) and replace *PS* with the output of an ML task (e.g., *parse tree* for the parsing task); therefore, in theory, it is possible that one can build a general-purpose, high-quality conversion algorithm, just like one can build a good statistical parser. On the other hand, while it is likely that a small number of (DS, PS) pairs exist for the language of our interest (e.g., trees for sentences in the annotation guidelines), we cannot assume that the number of pairs would be very large (say tens or hundreds of thousand pairs) because if there are so many (DS, PS) pairs available, DS-to-PS conversion is no longer important as one can easily create a PS treebank from these pairs. Just like there does not exist a general-purpose parser that performs well when trained on a few hundreds of (sentence, parse tree) pairs, we doubt that there exists a general-purpose DS-to-PS conversion algorithm that performs well when *trained* on a few hundreds (DS, PS) pairs, because, in the worst scenario, the analyses chosen by the DS and PS guidelines for linguistic phenomena can be so different that building PS from a given DS is not much easier than building a parse tree from a sentence.

Instead, we believe that high-quality DS-to-PS conversion is possible only if all of the following conditions hold: (1) the analyses chosen by DS and PS guidelines for most linguistic phenomena are compatible; (2) for the phenomena with incompatible analyses, the incompatibility can be resolved by simple transformations; and (3) for phenomena that are represented in the PS but not in the DS, the additional information needed to build PS is available from the PB or other sources.

If these conditions hold, high-quality DS-to-PS conversion is possible. Our conversion process for creating a PS from DS plus PB is illustrated in Figure 2.[2] It has two main modules: the first module handles phenomena with incompatible DS/PS analyses or phenomena represented only in the PS analyses. The input are DS and PB, and the output is a new, "extended" dependency structure called *DS+*. *DS+* should be consistent with the desired PS according to the PS guidelines.

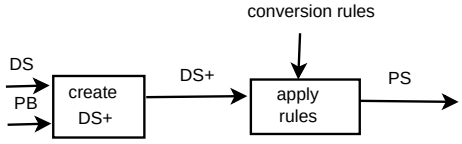


Figure 2: Building PS from DS and PB

The second module creates the PS from *DS+* by applying *conversion rules*. The *conversion rule* is a (DS-pattern, PS-pattern) pair, which says the DS pattern in a DS would correspond to the PS pattern in a PS tree. Figure 3 shows two conversion rules that will be used to create the PS in (1d) from the DS in (1b). The first rule says that when a verb in a DS has a left *k1* dependent whose head is a noun, the corresponding PS will have a *VP* node, which has an *NP* child followed by a *VP-Pred* child. The second rule is interpreted similarly. The conversion rules can be created by hand or extracted from consistent (DS, PS) pairs. The formal definition of conversion rules and the algorithms for extracting rules from (DS,PS) pairs and applying rules to generate a PS were discussed in (Xia et al., 2009).

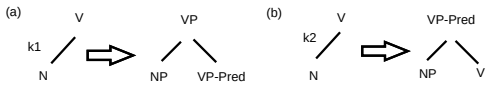


Figure 3: Two conversion rules

4. Handling incompatibility

As discussed in the previous section, we propose to use DS+ to handle phenomena with incompatible DS and PS analyses. The question is what DS+ should look like and how it can be created from the input DS and PB. In order to answer the question, we first need to understand the main sources of incompatibility. We will then go over seven linguistic phenomena that have

[1] The *flatten* operation merges all the internal nodes in the PS with their head child; the *label replacement* operation replaces the label of an internal node with its head word.

[2] This corresponds to the test stage in Figure 1.

incompatible analyses in the HUTB and show the corresponding DS+.

4.1. Main sources of incompatibility

Given that DS and PS guidelines are often based on different linguistic theories, there can be many reasons for incompatibility between the DS and PS analyses. Some instances of incompatibility could be accidental in that the DS and the PS might just choose distinct analyses even though in principle they could have picked the same analysis. Since we have developed the DS and PS guidelines in tandem, we have attempted to minimize the accidental incompatibilities. That leaves us with the more deep-seated sources of incompatibility. Here, we discuss three main reasons that cover the majority of such incompatibility in the HUTB.

The first reason is that one side chooses to represent certain relationships or distinctions, but the other does not. One example is the unaccusative vs. unergative distinction, which is represented in PS, not in DS. In order to create the desired PS, the list of unaccusative verbs has to be available from other sources, and in the HUTB that information comes from the PB.

The second reason is due to different representational vehicles that are available in DS and PS. In the HUTB, DS represents information through structural means, dependency labels (e.g., k1 and k2), or attributes in the nodes. PS represents information through structural means, syntactic labels (e.g., *NP*), and coindexation (e.g., between a trace and its antecedent). Consequently, the DS and the PS could represent the same information, but through different vehicles. The corresponding DS and PS trees could end up being inconsistent, because the definitions of consistency and compatibility look at tree structure only. In the HUTB, the analyses for passive, small clause, support verb, and causative fall into this category.

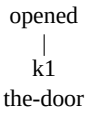
The third reason is due to the differences in handling word order by the DS and the PS.[3] The DS in the HUTB allows for non-projective trees and it does not have a notion of canonical word order. In contrast, the PS tree in the HUTB must be projective and it assumes that the core arguments are generated in distinguished structural positions which implies that there is an inherent notion of canonical word order. Consequently any DS trees that are non-projective or in which core arguments appear in non-canonical word order would be inconsistent with the corresponding PS trees.

4.2. Unaccusatives vs. Unergatives

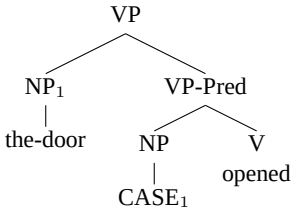
In the HUTB, the DS treats all intransitives alike while the PS makes a structural distinction between unergatives and unaccusatives: the PS treats the subject of an unaccusative as originating in the object position, as indicated by the empty category *CASE* and the coindexation between the subject and the object positions; the PS treats the subject of an unergative as originating in the subject position and there is no movement involved. Two examples are given in Ex (2) and (3).

(2) unaccusative: The door opened.

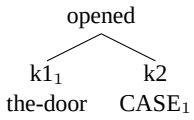
a. DS tree:



b. PS Tree:

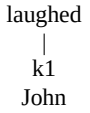


c. DS+ tree:

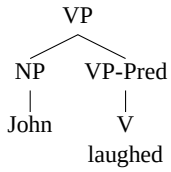


(3) Unergative: John laughed.

a. DS tree:



b. PS Tree:



For automatic conversion to be a possibility, information about whether a given verb is unergative or unaccusative needs to be available. In the HUTB, that information is provided in the PB. The next question is what DS+ looks like. One intuition is that DS+ should include all the empty categories (ECs) appearing in the PS. In this case, we need to insert *CASE* to the DS+. Based on the DS and PS trees in Ex (2), *V* is the head child of *VP-Pred*; therefore, *CASE* should depend on *opened* in the DS+. As for its dependency type, *CASE* is in the canonical object position in the PS, and the dependency type for that position is *k2* in general, as shown in the second rule in Figure 3. Therefore, we will insert *CASE* as a dependent of *opened* with the type *k2*, and we use coindexation to link the EC and its antecedent. The resulting tree is in Ex (2c). In contrast, unergatives do not require DS+ (that is, its DS+ is the same as DS).

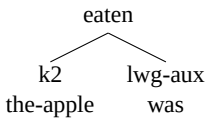
[3] This can be seen as a special case of the first reason; that is, the PS represents word order, whereas DS does not. But because word order is so salient and common, we treat this as a separate case.

4.3. Passive

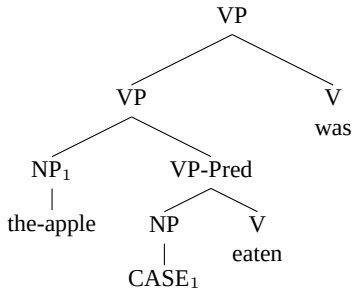
In the HUTB, both DS and PS indicate that the subject of the passive is related to the object position: the DS uses dependency type *k2*, and the PS uses the EC *CASE* and the coindexation between the subject and the object positions, as shown in Ex (4).[4]

(4) The apple eaten was ('The apple was eaten')

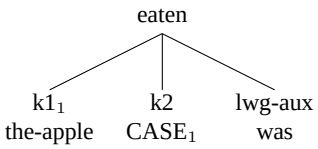
a. DS tree:



b. PS Tree:



c. DS+ Tree:



To detect passive is easy because a passive verb in the DS has a feature *passive='+'*. The DS+ for passive is similar to the one for unaccusative except that we change the dependency type of *the apple* from *k2* to *k1* because the phrase is in the canonical subject position in the PS, not the canonical object position.

4.4. Small clause and support verb

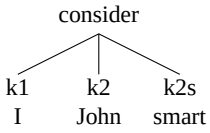
The two phenomena we have discussed so far involve only one predicate (the unaccusative verb or the passivized verb) in both DS and PS. Small clauses are different in that they involve two predicates, as shown in Ex (5): *consider* and *smart*. *John* is related to both predicates: it gets case from *consider* and semantically it is an argument of *smart*.

Both the DS and the PS represent these relations, but they do so in different ways. The DS represents the first relation by making *John* a dependent of *consider*; it represents the second relation by using the dependency type *k2s* for *smart*, and *k2s* encodes the information that its semantic argument has the label *k2*. The PS represents the two relations by inserting an EC *CASE* and coindexing it with *John*, and thus represents both relations structurally. Creating DS+ is simple; we just need to insert an EC *CASE* as a *k1* dependent of *smart* and coindex it with *John*.

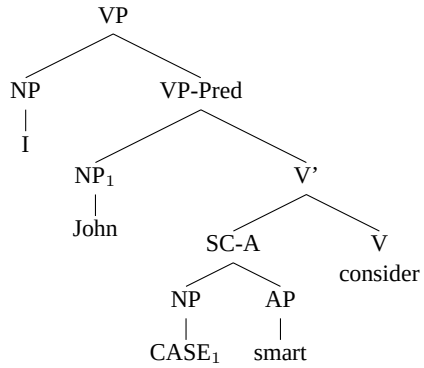
[4]The dependency type *lwg-aux* indicates that *was* is an auxiliary verb, and the word and its head *eaten* form a local word group (*lwg*)

(5) I John smart consider ('I consider John smart')

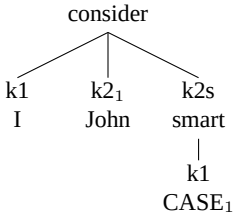
a. DS tree:



b. PS tree:



c. DS+ tree:



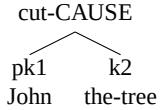
In the support verb construction, a verb and a noun form a complex predicate. An example is *John bicycle theft did* ('John stole a bicyle'), where *did* and *theft* form a complex predicate. Our treatment of support verb is similar to that of small clauses.

4.5. Causatives

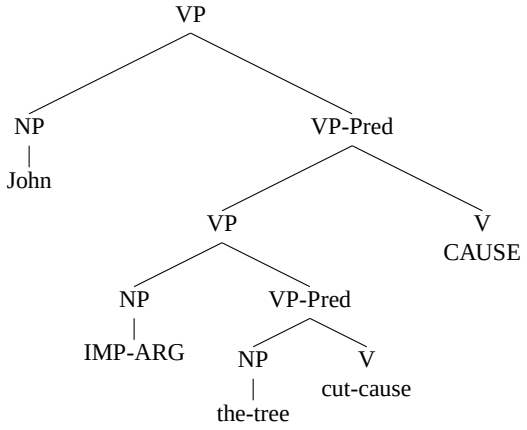
Causative is another example where DS and PS represent the same information through different means. An example is given in Ex (6). The DS analyzes the causativized verb as a single head, but it labels the causer *John* as *pk1* (not *k1*), indicating that *John* is not really an argument of *cut*, but an argument of the causative part of *cut-CAUSE*. The PS represents the causativized verb as two independent heads: an EC, *CAUSE*, as the head of the higher clause, and the original verb as the head of the lower clause. In addition, the PS indicates the implicit intermediate agent explicitly as an EC, *IMP-ARG* (implicit argument).

(6) John the-tree cut-CAUSE ('John caused the tree to be cut'.)

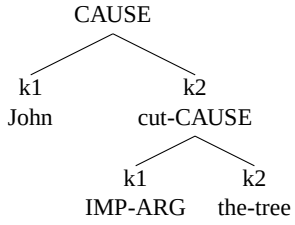
a. DS tree:



b. PS tree:



c. DS+ tree:



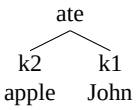
To create DS+, we insert two ECs: *CAUSE*, as the head of the higher clause, and another EC, *IMP-ARG* as a *k1* dependent of the lower clause. Furthermore, the causee becomes a dependent of *CAUSE* and its label is changed from *pk1* to *k1*.

4.6. Movement

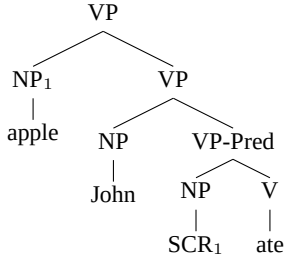
The last type of divergences involve the treatment of movement. In HUTB, the DS is not concerned about non-canonical word order; sentences with different word orders will have the same DS if we treat the DS as an unordered tree. In contrast, the PS assumes that the dependents of a head have a canonical order and if they are not in the canonical order, that is due to syntactic movement which is represented by an EC in the base position and a coindex between the base position and the surface position. An example is in Ex (7).

(7) apple John ate ('John ate an apple')

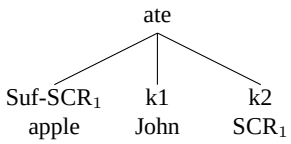
a. DS tree:



b. PS tree:



c. DS+

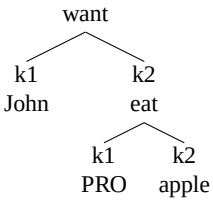


To create DS+ for this example, we need to know the canonical order of arguments of a verb. In Hindi, the order is *k1, k4, k2, verb*. By checking the word order in the sentence, we can detect the predicates whose dependents are not in the canonical order. We then use simple heuristics to determine which dependent is *moved* (in this case, it is *k2*). Next, we insert an EC *SCR* to DS+ as a *k2* dependent of the verb, replace the label of *apple* from *k2* to a new dependency type *Suf-SCR* (for the surface position of a scrambled element), and coindex the EC with *apple*.

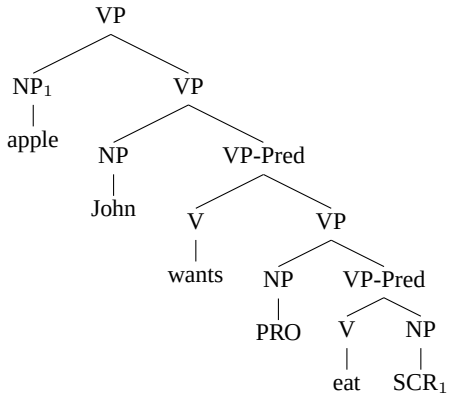
Movement in Ex (7) does not cause non-projectivity, because it does not cross the boundary of the clause. When movement crosses a clause boundary, its DS tree will be non-projective, see Ex (8).

(8) apple, John eat want ('John wants to eat an apple')

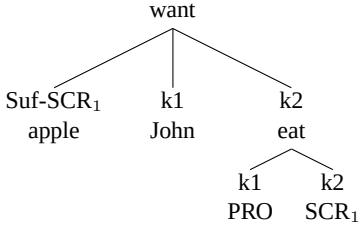
a. DS tree:[5]



b. PS tree:



c. DS+ tree:



Detecting non-projectivity is trivial given the original sentence and the DS. The creation of DS+ is similar to the process for the local movement, except that the

[5]The EC, PRO, is actually added by the PB.

moved element (*apple* in this example) will be moved up along the path from its parent to the root of the DS until its new position resolves the non-projectivity. In this example, *apple* becomes a child of *want* in DS+. Its dependency relation to *eat* is implicit as its trace *SCR* depends on *eat*.

Most movement in Hindi is to the left, as in Ex (7) and (8). But movement to the right is possible. The creation of DS+ for rightward movement is not discussed here due to the limitation of space.

4.7. Combinations

So far we have discussed seven phenomena where DS+ is needed to bridge the differences between DS and PS analyses. For each phenomenon, we have created a *rule* (i.e., a piece of code) that detects the phenomenon in a given DS plus PB and builds the DS+ accordingly.

Some of these phenomena can co-occur to the same predicate and its dependents in a DS; for instance, the arguments of a causative verb can undergo leftward or rightward movement ('a book John caused Mary to be given'); the main verb in a small clause construction can be passivized (e.g., 'John is considered intelligent'). We call them *combinations* of phenomena. The question is whether we can handle such combinations without writing more rules. In other words, (1) what combinations of phenomena are possible? (2) For these combinations, can the correct DS+ be created by applying the rules for individual phenomena in a certain order? (3) If so, what should the order be?

For the first question, some combinations are impossible. For instance, an unaccusative verb (e.g., *break* in 'window broke') lacks an external argument and cannot be passivized, so unaccusative + passive does not exist. Based on our observations on grammaticality of various combinations, we group the seven phenomena into five groups so that all the possible combinations consist of at most one phenomenon from each group:

- Group 1: unaccusative, passive
- Group 2: small clause, support verb
- Group 3: causative
- Group 4: rightward movement
- Group 5: leftward movement

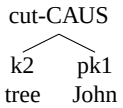
We show that the answer to the second question is *yes* by using the ordering based on the five groups; that is, applying the two rules in Group 1 first, followed by the rules for Group 2, 3, 4, and 5. This is the same order if we sort the rules based on the size of the region affected by the rules: Rules in Group 1 only affect a simple clause; rules in Group 2 affect a clause that contains a small clause; the rule in Group 3 affects a higher clause and a lower clause; the rules in Groups 4 and 5 can affect multiple clauses as movement can cross clause boundaries.[6]

[6]Note that not all the orderings would yield the desired PS. For instance, leftward movement (group 5) should be handled after rightward movement (group 4) because in the DS+ for rightward movement, a trace for the moved element needs to be immediately before the verb, which might not be the canonical position for that element and this can trigger leftward movement.

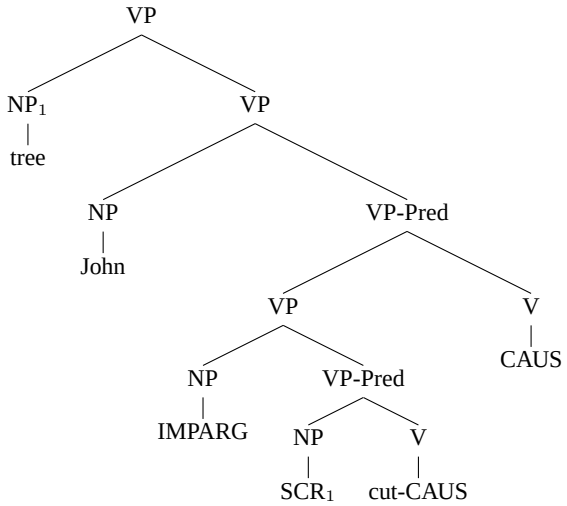
Now that we have fixed the ordering of the rules, we test whether applying rules in that order would produce the desired DS+. It turns out that the answer is indeed affirmative. Due to the limitation of space, we will just show an example. In (9), (b) and (c) are the input DS and the desired PS, respectively; (d) is the resulting DS+ after applying the rule for causative to (b); (e) is the resulting DS+ after applying the rule for leftward movement to (d), and it is indeed the DS+ that we want to create and it is consistent with the PS.

(9) a. tree John cut-CAUS ('John caused the tree to be cut')

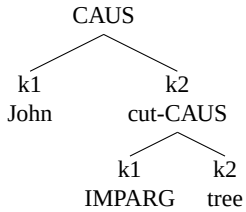
b. DS:



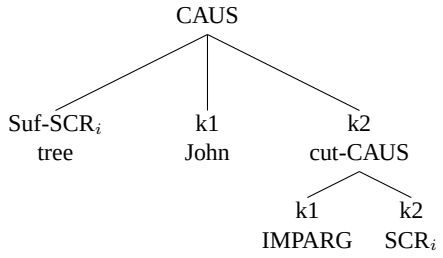
c. PS:



d. DS+ after applying the causative rule to (b):



e. DS+ after applying the leftward movement rule to (d):



5. Discussion

The previous section went over seven phenomena for which DS and PS analyses are incompatible. Due to the limit of space, we used very simple examples and did not explain all the details. These details mean that the step of creating DS+ can be very complex; it requires manually going through all the phenomena with incompatible DS and PS analyses, and for each phenomenon determining what are the diagnostic tests for detecting the phenomenon and what DS+ should look like, and then writing rules to build the DS+. Furthermore, one needs to check whether or not the combinations of phenomena can be handled by applying rules in a particular order. Now we are ready to address the questions raised in Section 1.

First, a general-purpose, high-quality DS-to-PS conversion algorithm is unlikely to exist, because the number of (DS, PS) pairs is too small for building a statistical system; consequently, any high-quality conversion would require manual comparison of DS and PS analyses for each linguistic phenomenon; this process is time consuming and cannot be fully automated.

Second, the DS and PS guideline designers have some freedom in choosing analyses for linguistic phenomena, because DS+ serves as a vehicle to bridge the gap between the DS and PS analyses. However, the bigger the gap is, the more complex the module for creating DS+ will be. Therefore, when DS and PS guideline designers choose incompatible analyses, the decisions should be well-motivated.

Third, as shown in Figure 2, the input to the conversion process are (1) a set of sentences with three layers of annotation, which is used for extracting conversion rules, (2) the sentences with DS and PB annotation for which PS will be created, (3) rules manually-crafted for creating DS+. In order to create the desired PS, any information needed to form the PS has to be available in (1), (2), or (3).

6. Conclusion

An important question for treebank development is whether high-quality conversion from one representation to another representation (e.g., PS) is possible, assuming that annotation guidelines exist for both representations. In this study, we focus on DS-to-PS conversion, and demonstrate that conversion is possible only when certain conditions are met. We propose to use DS+ as a vehicle to bridge the gap between DS and PS analyses. When these conditions are met, PS can be created in two steps: creating DS+ from the input DS plus PB and generating PS from DS+. We then go over seven phenomena in the HUTB for which DS+ is needed, and show that creating DS+ is complex and cannot be fully automated. For future work, we will test our conversion process on the HUTB and evaluate the system performance on a small portion of the treebank where all three layers are manually annotated.

Acknowledgments This work is supported by NSF grants 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. Special thanks go to Annahita Farudi and Michael Tepper.

7. References

Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai, and Rajeev Sangal. 2008. Dependency annotation scheme for indian languages. In Proceedings of IJCNLP, Hyderabad, India.

Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1995. Natural Language Processing – A Paninian Perspective. Prentice-Hall of India.

Rajesh Bhatt, Owen Rambow, and Fei Xia. 2011. Linguistic phenomena, analyses, and representations: Understanding conversion between treebanks. In Proceedings of IJCNLP, pages 1234–1242, Chiang Mai, Thailand.

Aoife Cahill, Mairead McCarthy, Josef van Genabith, and Andy Way. 2002. Automatic Annotation of the Penn-Treebank with LFG F-Structure Information. In LREC 2002 Workshop on Linguistic Knowledge Acquisition and Representation - Bootstrapping Annotated Language Data.

Noam Chomsky. 1981. Lectures on Government and Binding. Dordrecht: Foris.

Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillmann. 1999. A statistical parser for czech. In Proceedings of ACL, pages 505–512.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. Computational Linguistics, 33(3):355–396.

Paul Kingsbury, Martha Palmer, and Mitch Marcus. 2002. Adding semantic annotation to the Penn TreeBank. In Proceedings of HLT, San Diego, CA.

Joakim Nivre. 2003. Theory-supporting treebanks. In In Proceedings of the TLT 2003 Workshop.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. Computational Linguistics, 31(1):71–106.

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, Hyderabad.

Fei Xia and Martha Palmer. 2001. Converting Dependency Structures to Phrase Structures. In Proceedings of HLT, San Diego, CA.

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.