Using Supertags and Encoded Annotation Principles for Improved Dependency to Phrase Structure Conversion

Seth Kulick and **Ann Bies** and **Justin Mott**
Linguistic Data Consortium
University of Pennsylvania
Philadelphia, PA 19104
{skulick,bies,jmott}@ldc.upenn.edu

Abstract

We investigate the problem of automatically converting from a dependency representation to a phrase structure representation, a key aspect of understanding the relationship between these two representations for NLP work. We implement a new approach to this problem, based on a small number of supertags, along with an encoding of some of the underlying principles of the Penn Treebank guidelines. The resulting system significantly outperforms previous work in such automatic conversion. We also achieve comparable results to a system using a phrase-structure parser for the conversion. A comparison with our system using either the part-of-speech tags or the supertags provides some indication of what the parser is contributing.

1 Introduction and Motivation

Recent years have seen a significant increase in interest in dependency treebanks and dependency parsing. Since the standard training and test set for English parsing is a phrase structure (PS) treebank, the Penn Treebank (PTB) (Marcus et al., 1993; Marcus et al., 1994), the usual approach is to convert this to a dependency structure (DS) treebank, by means of various heuristics for identifying heads in a PS tree. The resulting DS representation is then used for training and parsing, with results reported on the DS representation.

Our goal in this paper is to go in the reverse direction, from the DS to PS representation, by finding a minimal DS representation from which we can use an approximate version of the principles of the PTB guidelines to reconstruct the PS. Work in this conversion direction is somewhat less studied (Xia et al., 2009; Xia and Palmer, 2001), but it is still an important topic for a number of reasons. First, because both DS and PS treebanks are of current interest, there is an increasing effort made to create multi-representational treebank resources with both DS and PS available from the beginning, without a loss of information in either direction (Xia et al., 2009). Second, it is sometimes the case that it is convenient to do annotation in a dependency representation (e.g., if the annotators are already familiar with such a representation), though the treebank will in final form be either phrase-structure or multi-representational (Xia et al., 2009).

However, our concern is somewhat different. We are specifically interested in experimenting with dependency parsing of Arabic as a step in the annotation of the Arabic Treebank, which is a phrase structure treebank (Maamouri et al., 2011). Although we currently use a phrase structure parser in this annotation pipeline, there are advantages to the flexibility of being able to experiment with advances in parsing technology for dependency parsing. We would like to parse with a dependency representation of the data, and then convert the parser output to a phrase structure representation so that it can feed into the annotation pipeline. Therefore, in order to make use of dependency parsers, we need a conversion from dependency to phrase structure with very high accuracy, which is the goal of this paper.

While one of our underlying concerns is DS to PS conversion for Arabic, we are first focusing on a conversion routine for the English PTB because it is so well-established and the results are easier to interpret. The intent is then to transfer this conversion algorithm work to the Arabic treebank as well. We expect this to be successful because the ATB has some fundamental similarities to the PTB in spite of the language difference (Maamouri and Bies, 2004).

As mentioned above, one goal in our DS to PS conversion work is to base it on a minimal DS representation. By "minimal", we mean that it does not include information that is redundant, together with our conversion code, with the implicit information in the dependency structure itself. As discussed more in Section 2.1, we aim to make our dependency representation simpler than "hybrid" representations such as Johansson and Nugues (2007). The reason for our interest in this minimal representation is parsing. We do not want to require the parser to recover such a complex dependency representations, when it is, in fact, unnecessary, as we believe our approach shows. The benefit of this approach can only be seen when this line of work is extended to experiments with parsing and Arabic conversion. The work described here is just the first step in this process.

A conversion scheme, such as ours, necessarily relies on some details of the annotation content in the DS and PS representations, and so our algorithm is not an algorithm designed to take as input any arbitrary DS representation. However, the fundamentals of our dependency representation are not radically different than others - e.g. we make an auxiliary verb the child of the main verb, instead of the other way, but such choices can be adjusted for in the conversion.

To evaluate the success of this conversion algorithm, we follow the same evaluation procedure as Xia et al. (2009) and Xia and Palmer (2001). We convert the PTB to a DS, and then use our algorithm to convert the DS back to a PS representation. The original PS and the converted-from-DS PS are then compared, in exactly the same way as parser output is compared with the original (gold) tree. We will show that our results in this area are a significant improvement above previous efforts.

A key aspect of this work is that our DS-to-PS conversion encodes many of the properties of the PTB annotation guidelines (Bies et al., 1995), both globally and for specific XP projections. The PTB guidelines are built upon broad decisions about PS representation that provide an overall framework and cohesion for the details of the PS trees. To implement these underlying principles of the guidelines, we defined a set of 30 "supertags" that indicate how a lexical item can project in the syntactic structure, allowing us to specify these principles. We describe these as supertags because of a conceptual similarity to the supertagging work in the Tree Adjoining Grammar (TAG) tradition (Bangalore and Joshi, 2010), although ours is far smaller than a typical supertag set, and indeed is actually smaller than the PTB POS tag set.

Our DS-to-PS code is based on this set of supertags, and can be run using either the supertags created from the gold POS tags, or using the POS tags, together with the dependency structure to first (imperfectly) derive the supertags, and then proceed with the conversion. This choice of starting point allows us to measure the impact of POS tag complexities on the DS-to-PS conversion, which provides an interesting insight on what a phrase structure parser contributes in addition to this sort of automated DS-to-PS conversion, as discussed in Section 4.

We have chosen this approach of encoding underlying principles of the PTB guidelines for two reasons. First, these principles are non-statistical, and thus we felt it would let us tease apart the contribution of the frequency information relating, e.g., heads, on the one hand, and the basic notions of phrase structure on the other. The second reason is that it was quite easy to implement these principles. We did not attempt a complete examination of every possible rule in Bies et al. (1995), but rather just selected the most obvious ones. As we will see in Section 4.2, our results indeed are sometimes hurt by such lack of thoroughness, although in future work we will make this more complete.

2 Overview and Example

Figures 1-4 provide a running example of the four steps in the process. Figure 1 is the original tree from the Penn Treebank. Figures 2 and 3 illustrate the two-step process of creating the dependency representation, and Figure 4 shows the conversion back to phrase structure.

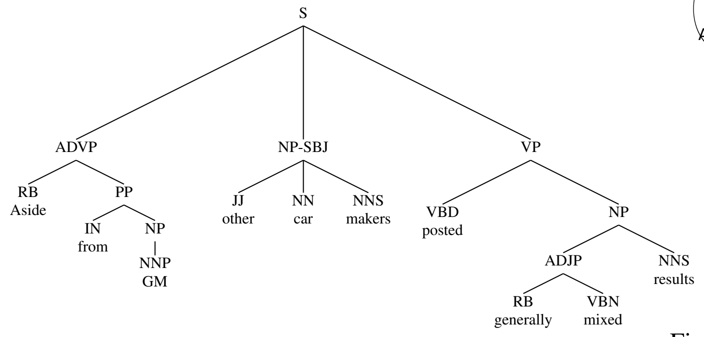


Figure 1: Penn Treebank tree

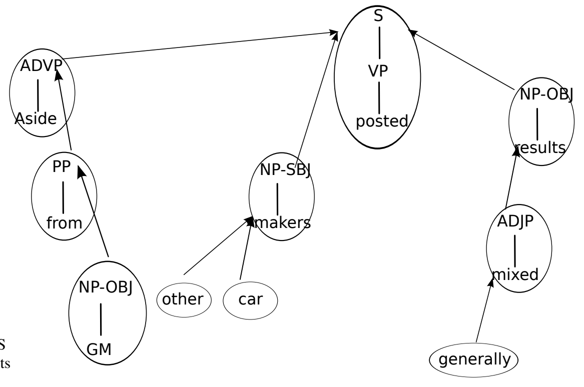


Figure 2: Tree Insertion Grammar decomposition of Figure 1

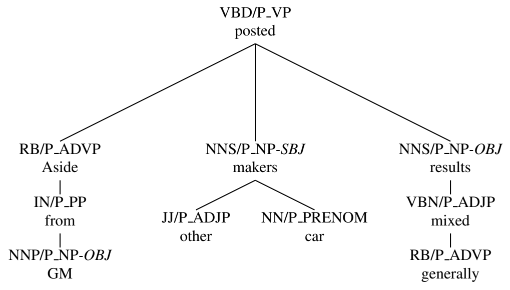


Figure 3: Dependency representation derived from TIG decomposition in Figure 2

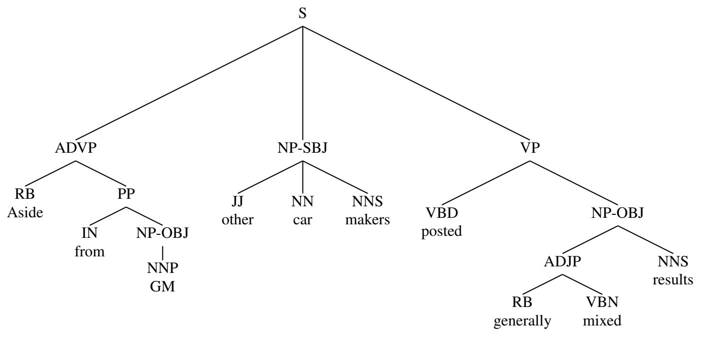


Figure 4: Conversion of dependency representation in Figure 3 back to phrase structure.

2.1 Creation of Dependency Representation

The creation of the dependency representation is similar in basic aspects to many other approaches, in that we utilize some basic assumptions about head relations to decompose the full tree into smaller units. However, we first decompose the original trees into a Tree Insertion Grammar representation (Chiang, 2003), utilizing tree substitution and sister adjunction. We refer the reader to Chiang (2003) for details of these operations, and instead focus on the fact that the TIG derivation tree in Figure 2 partitions the phrase structure representation in Figure 1 into smaller units, called elementary trees. We leave out the POS tags in Figure 2 to avoid clutter.

The creation of the dependency representation is structurally a simple rewrite of the TIG derivation, taking the word associated with each elementary tree and using it as a node in the dependency tree. In this way, the dependency representation in Figure 3 follows immediately from Figure 2.

However, in addition, we utilize the TIG derivation tree and the structures of the elementary trees to create a supertag (in the sense discussed in Section 1) for each word. For example, `aside` heads an elementary tree that projects to `ADVP`, so it is assigned the supertag P_ADVP in Figure 3, meaning that it projects to ADVP. We label each node in Figure 3 with both its POS tag and supertag, so in this case the node for `aside` has RB/P_ADVP.

There are two typical cases that are not so straightforward. The first concerns elementary trees with more than one level of projection, such as that for the verb, `posted`, which has two levels of projection, S and VP. In such cases we base the supertag only on the immediate parent of the word. For example, in this case the supertag for `posted` is P_VP, rather than P_S. As will be seen in Section 3.2, our perspective is that the local context of the dependency tree will provide the necessary disambiguation as to what node is above the VP.

Projection Type	Supertag
NP	P_NP
ADJP	P_ADJP
ADVP	P_ADVP
PP	P_PP, P_WHPP
S,SINV,SQ	P_VP
QP,NP,QP-NP,QP-ADJP	P_QP
WHNP	P_WHNP
default	P_WHADVP, P_INTJ, P_PRT, P_LST
none	P_AUX, P_PRENOM, P_DET, P_COMMA, P_PERIOD, P_CC, P_COMP, P_POS, P_PRP$, P_BACKDQUOTE, P_DQUOTE, P_COLON, P_DOLLAR, P_LRB, P_RB, P_PDT, P_SYM, P_FW, P_POUND

Table 1: 30 supertags handled by 14 projection types. The ambiguity in some, such as P_VP projecting as S, SINV, SQ is handled by an examination of the dependency structure.

The second non-straightforward case[1] is that of degenerate elementary trees, in which the "tree" is just the word itself, as for `other`, `car`, and `generally`. In such cases we default the supertag based on the original POS tag, and in some cases, the tree configuration. For example, a word with the JJ tag, such as `other`, would get the supertag P_ADJP, with the RB tag such as `generally` the supertag P_ADVP. We assign prenominal nouns such as `car` here the tag P_PRENOM.

Generating supertags in this way is a convenient way to correct some of the POS tag errors in the PTB (Manning, 2011). For example, if `that` has the (incorrect) tag DT in the complementizer position, it still receives the new POS tag `P_COMP`.

This procedure results in a set of 30 supertags, and Table 1 shows how they are partitioned into 14 projection types. These supertags and projection types are the basis of our DS-to-PS conversion, as discussed further in Section 2.2.

We note here a brief comparison with earlier work on "hybrid" representations, which encode a PS representation inside a DS one, in order to convert from the latter to the former. (Hall and Nivre, 2008; Johan Hall and Nilsson, 2007; Johansson and Nugues, 2007). Our goal is very different. Instead of encoding the phrase structure in the dependency tree via complex tags such as SBARQ in Johansson and Nugues (2007), we use a minimal representation and rely on our encoding of the general principles of PTB phrase structure to carry much of the weight. While supertags such as P_VP may appear to encode some of the structure, their primary role is as an intermediate link between the POS tags and the phrase structure conversion. The created supertags are not in fact necessary for this conversion. As we will see in the following sections, we convert from DS to PS using either just the original POS tags, or with our created supertags.

We also include five labels in the dependency representation: SBJ, OBJ, PRN, COORD_CONJ, APP. The example dependency tree in Figure 3 includes instances of the SBJ and OBJ labels, in italics on the node instead of the edges, for convenience. The SBJ label is of course already a function tag in the PTB. We process the PTB when creating the TIG decomposition to add an OBJ tag, as well basing the PRN label on the occurrence of the PRN node. We also use heuristics to identify cases of coordination and apposition, resulting in the COORD_CONJ and APP tags. The reasons for including these labels is that they prove useful in the conversion to phrase structure, as illustrated in some of the examples below.

Before moving on to the dependency-to-phrase-stucture conversion, we end this section with a comment on the role of function tags and empty categories. The PTB makes use of function tags to in-

[1]There are other details not discussed here. For example, we do not automatically assign a P_NP supertag to the head child of an NP, since such a head can legitimately be, e.g, a JJ, in which case we make the supertag P_ADJP, on the reasoning that it would be encoding "too much" to treat it as P_NP. Instead, we rely on the DS and such labels as SBJ or OBJ to determine when to project it as NP in the converted PS.

dicate certain syntactic and semantic information, and of empty categories (and co-indexing) for a more complete and accurate syntactic representation. There is some overlap between the five labels we use, as just described, and the PTB function tags, but in general we do not encode the full range of function tags in our representation, saving this for future work. More significantly, we also do not include empty categories and associated co-indexing, which has the consequence that the dependency trees are projective.

The reason we have not included these aspects in our representation and conversion yet is that we are focused here first on the evaluation for comparison with previous work, and the basis for this previous work is the usual evalb program (Sekine and Collins, 2008), which ignores function tags and empty categories. We return to this issue in the conclusion.

2.2 From Dependency to Phrase Structure

There are two key aspects to the conversion from dependency to phrase structure. (1) We encode general conventions about annotation that are used throughout the annotation guidelines for the PTB. A common example is that of the "single-word" rule, in which a constituent consisting of just a single word is reduced to just that word, without the constituent bracketing, in many cases. (2) We use the set of supertags as the basis for defining projection-specific rules for how to attach children on the left or right of the head, in many cases utilizing the supertag names that we include to determine the specific attachment.

For example, the leaf `GM` in Figure 3 has the supertag P_NP (with the label OBJ), so heading a NP projection, `(NP GM)`. Its parent node, `from`, has the supertag P_PP, indicating that it heads a PP projection, and so attaches the `(NP GM)` as a sister of `from`. It does not reduce it down as a single word, because the encoding of the PP projection specifies that it does not do so for children on its right.

A more substantial case is that of the NP `other car makers`. Here the head noun, `makers`, has the supertag P_NP, and so projects as an NP. Its first child, `other`, has the supertag P_ADJP, and so projects as an ADJP, resulting in `(ADJP other)`. The second child, `car`, has the supertag P_PRENOM (prenominal), and so does not project at all. When the NP projection for `makers` is assembled, it applies the "single-word" constraint to children on its left (as encoded in the definition of the NP projection), thus stripping the ADJP off of `other`, resulting in the desired flat NP `other car makers`. Likewise, the ADVP projection for `generally` is stripped off before it is attached as a left sister of the ADJP projection `mixed`. The encoding of a VP projection specifies that it must project above VP if it is the root of the tree, and so the VP projection for `posted` projects to S (by default).

In this way we can see that encoding some of the general characteristics of the annotation guidelines allows the particular details of the PTB phrase-structure representation to be created from the less-specific dependency representation.

3 Some Further Examples

3.1 QP Projection or Reduction

As mentioned in Section 2.2, the "single word" convention is implemented in the conversion to PS, as was the case with `other` in the previous section. The projection associated with P_QP has a slight twist to this principle, because of the nature of some of the financialspeak in the PTB. In particular, the dollar sign is treated as a displaced word and is therefore not counted, in a QP constituent, as a token for purposes of the "single token" rule.

For example, (1abc) in Figure 5 illustrates a case where the QP structure projects to an NP node as well. (1a) is the original PTB PS tree, and (1b) is the DS representation. Note that `billion` heads the `about $ 9 billion` subtree, with the supertag P_QP and the label OBJ.[2] Because it has more than one child in addition to the $, it is converted to phrase structure as a QP under an NP, implying the empty *U*, although we do not actually put it in.

In contrast, (2abc) is a case in which the QP node is not generated. 100 is the head of the phrase $ 100 *U* in the PTB PS (a), as shown in the dependency structure (b). However, because it only has one child in addition to the $, no additional QP node is created in the phrase structure representation in (c). We stress that the presence of the QP in (1a) and its ab-

[2] A good case can be made that in fact $ should be the daughter of `to` in the dependency tree, although we have not implemented this as such.

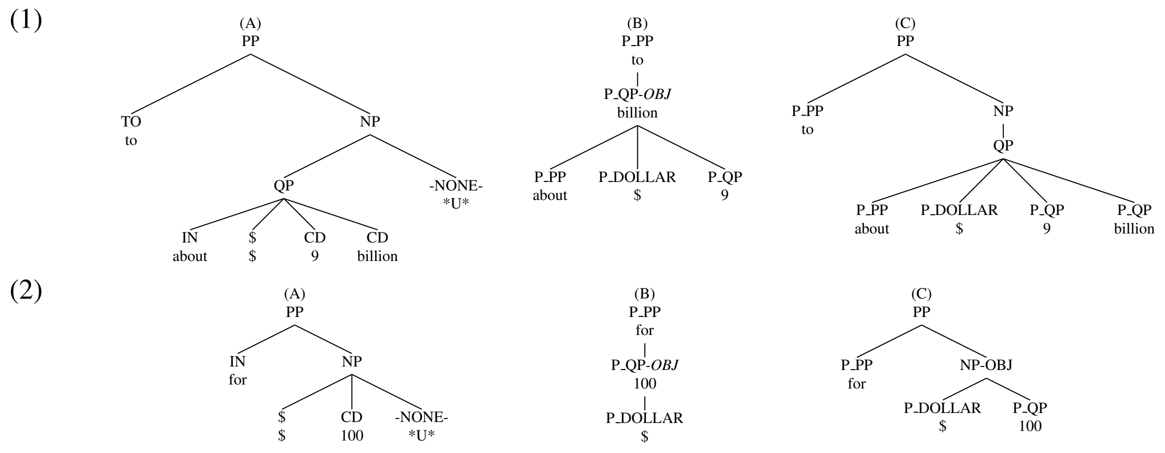


Figure 5: Examples of handling of QP in dependency to phrase-structure conversion.

sence in (2a) is correct annotation, consistent with the annotation guidelines.

3.2 Refinement of VP Projections

As mentioned above, instead of having separate supertags for S, SINV, SQ, SBAR, SBARQ, we use only the P_VP supertag and let the context determine the specifics of the projection. Sentences (3ab) in Figure 6 illustrate how the SBJ label is used to treat the P_VP supertag as indicating projection to SINV (or SQ) instead of S. The determination is based on the children of the P_VP node. For example, if there is a child with the P_AUX supertag which is before a child with the SBJ label, which in turn is before the P_VP node itself, then the latter is treated as projecting to either SINV or SQ, depending on the some additional factors, primarily whether there is a WH word among the children. In this example, there is no WH word, so it becomes a SINV.[3] We note here that we also include a simple listing of verbs that take complements of certain types - such as verbs of saying, etc., that take SBAR complements, so that a VP will project not just to S, but SBAR, even if the complement is missing.

3.3 Coordination

We represent coordination in the dependency in one of the standard ways, by making the following conjuncts be children of the head word of

[3]This is not a fully precise implementation of the conditions distinguishing SQ and SINV projections, in that it does not properly check for whether the clause is a question.

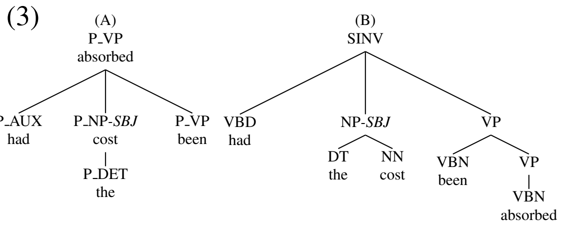


Figure 6: (3ab) shows that the local context of the P_VP supertag in the dependency tree results in a SINV structure in the converted phrase structure tree (3b).

the first conjunct. For example, a dependency representation of `...turn down the volume and close the curtains` is shown in (4a) in Figure 7. The conjunct `close the curtains` is converted as a VP projection projecting to S. However, when the projection for `turn` is assembled, the code checks if the conjuncts are missing subjects, and if so, reduces the configuration to standard VP coordination, as in (4b). The COORD label is used to identify such structures for examination.

4 Results of Dependency to Phrase Structure Conversion

To evaluate the correctness of conversion from dependency to phrase structure, we follow the same strategy as Xia and Palmer (2001) and Xia et al. (2009). We convert the phrase structure trees in the PTB to dependency structure and convert the dependency back to phrase structure. We then compare the original PTB trees with the newly-created phrase

(4)

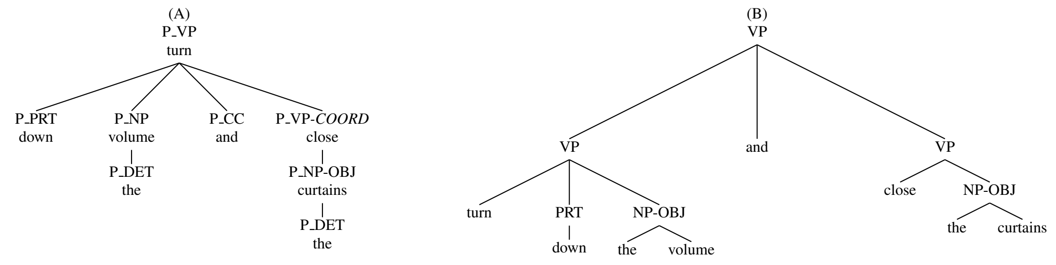


Figure 7: (4a) is the dependency representation of a coordination structure, and the resulting phrase structure (4b) shows that the conversion treated it as VP coordination, due to the absence of a subject.

Sec	System	rec	prec	f
00	Xia & Palmer '01	86.2	88.7	87.5
	Xia et al. '09	91.8	89.2	90.5
	USE-POS-UNLABEL	96.6	97.4	97.0
	USE-POS	94.6	95.4	95.0
	USE-SUPER	95.9	97.0	96.4
22	Xia et al. '09	90.7	88.1	89.4
	USE-POS	95.0	95.5	95.3
	USE-SUPER	96.4	97.1	96.7
23	Wang & Zong '10	95.9	96.3	96.1
	USE-POS	94.8	95.7	95.3
	USE-SUPER	96.2	97.3	96.7
24	USE-POS	94.0	94.7	94.4
	USE-SUPER	95.9	97.1	96.5

Table 2: Results of dependency to phrase structure conversion. For our system, the results are presented in two ways, using either the gold part-of-speech tags (USE-POS) or our gold supertags (USE-SUPER). For purposes of comparison with Xia and Palmer (2001) and Xia et al. (2009), we also present the results for Section 00 using part-of-speech tags, but with an unlabeled evaluation (USE-POS-UNLABEL).

structure trees, using the standard evalb scoring code (Sekine and Collins, 2008). Xia and Palmer (2001) defined three different algorithms for the conversion, utilizing different heuristics for how to build projection chains, and where to attach dependent subtrees. They reported results for their system for Section 00 of the PTB, and we include in Table 2 only their highest scoring algorithm. The system of Xia et al. (2009) uses conversion rules learned from Section 19, and then tested on Sections 00 and Section 22.

We developed the algorithm using Section 24, and we also report results for Sections 00, 22, and 23, for comparison with previous work. We ran our system in two ways. In one we use the "gold" supertags that were created as described in Section 2.1 (USE-SUPER), based on the TIG decomposition of the original tree. In the other (USE-POS) we use the gold POS tags, and not the supertags. Because our DS-to-PS algorithm is based on using the supertags to guide the conversion, the USE-POS runs work by using a few straightforward heuristics to guess the correct supertag from the POS tag and the dependency structure. For example, if a word x has the POS tag "TO" and the word y to its immediate right is its parent in the dependency tree and y has one of the verbal POS tags, then x receives the supertag P_AUX, and otherwise P_PP. Any word with the POS tag JJ, JJR, or JJS, receives the supertag P_ADJP, and so on. The results for Xia and Palmer (2001) and Xia et al. (2009) were reported using an unlabeled version of evalb, so to compare properly we also report our results for Section 00 using an unlabeled evaluation of the run using the POS tags (USE-POS-UNLABEL), while all the other results use a labeled evaluation.

We also compare our system with that of Wang and Zong (2010). Unlike the three other systems (including ours), this was not based on an automatic conversion from a gold dependency tree to phrase structure, but rather used the gold dependency tree as additional input for a phrase structure parser (the Berkeley parser).

4.1 Analysis

While our system was developed using Section 24, the f-measure results for USE-SUPER are virtually identical across all four sections (96.4, 96.7, 96.7, 96.5). Interestingly, there is more variation in the

USE-POS results (95.0, 95.3, 95.3, 94.4). We take this to be an indication of a difference in the sections as to the utility of the POS tags to "bootstrap" the syntactic structure. As just mentioned above, the USE-POS runs work by using heuristics to approximate the gold supertags from the POS tags.

The supertags, because they are partially derived from the phrase structure, can obscure a disconnect between a POS tag and the syntactic structure it projects. For example, the word `according` in the structure `(PP (VBG according) (PP (TO to) ...))` receives the gold supertag P_PP, a more explicit representation of the word's role in the structure than the ambiguous VBG. This is why the USE-POS score is lower than the USE-SUPER score, since the POS tag and dependency structure do not always, at least with our simple heuristics, lead to the gold supertag. For example, in the USE-POS run, `according` receives the incorrect supertag P_VP, leading to an incorrect structure, while in the USE-SUPER run, it is able to use P_PP, leading to the correct structure.

However, even with the lower performance of USE-POS, it is well above the results reported in Xia et al. (2009) for Section 22, and even more so with the unlabeled evaluation of Section 00 compared to Xia and Palmer (2001) and Xia et al. (2009). The comparison with Wang and Zong (2010) for Section 23 (they did not report results for any other section) shows something very different, however. Their result, using a gold dependency tree together with the Berkeley parser, is above our USE-POS version and below our USE-SUPER version.

Our interpretation of this is that it provides an indication of what the parser is providing on top of the gold dependency structure, which is roughly the same information that we have encoded in our DS to PS code. However, because the Wang and Zong (2010) system performs better than our USE-POS version, it is likely learning some of the non-straightforward cases of how USE-POS tags can bootstrap the syntactic structure that our USE-POS version is missing. However, any conclusions must be tentative since our dependency structures are not necessarily the same as theirs and so it is not an exact comparison.

Error type	count
problem with PTB annotation	8
ambiguous ADVP placement	3
incorrect use of "single token rule"	3
FRAG/X	2
multiple levels of recursion	2
other	5

Table 3: Analysis of errors in first 50 sentences of USE-SUPER run for Section 24

4.2 Errors from Dependency Structure with Supertags to Phrase Structure

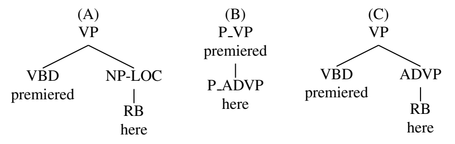
We stressed in the introduction that we are interested in understanding better the relationship between the DS and PS representations. Identifying areas where the conversion from DS did not result in a perfect (evalb score) PS is therefore of particular interest.

For this analysis, we used our dev section, 24, with the run USE-SUPER. We use this run because we are interested in cases where, even with the gold supertags, there was still a problem with the conversion to the PS. We examined the first 50 sentences in the section, with a total of 23 errors. We recognize that this is a very small sample. An eyeball examination of other sentences does not reveal anything significantly different than what we present here as far as the sorts of errors, although we have only performed a rigorous analysis of these 23 errors, which is why we limit our discussion here to these cases.

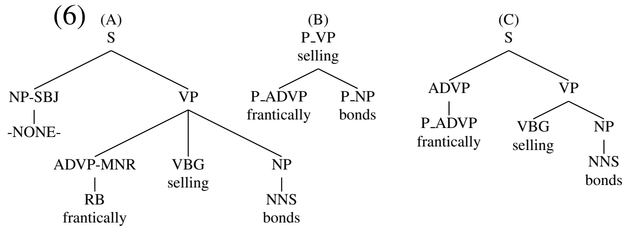
Table 3 shows a breakdown of these 23 errors. Note that by "error" here we mean a difference between the reconstructed PS structure, and the PTB gold PS structure, causing the score for Section 24, USE-SUPER (last row) in Table 2 to be less than perfect.

The most common "error" is that in which the PTB annotation is itself in error, while our algorithm actually creates a correct phrase structure, in the sense that it is consistent with the PTB guidelines. Three of these eight annotation problems are of the same type, in which a NP is headed by a word with the RB tag. An example is shown in (5) in which (5a) shows (a fragment of) the original tree in the PTB, and (5b) is the resulting DS, with (5c) the reconstructed PS tree. The word `here` receives the supertag P_ADVP, thus resulting in a different reconstructed PS, with an ADVP. There is a mismatch between the POS tag and the node label in the original tree (5a), and in fact in this case the node label in the PTB tree should have been ADVP-LOC, instead of NP-LOC.

(5)



(6)



An example of the "ambiguous ADVP placement" error is shown in (6), in which the PTB tree has the adverb `frantically` inside the VP, information which is not available in the DS (6b). Our conversion code has to choose as to where to put such ADVPs, and it puts them outside the VP, as in (6c), which is sometimes correct, but not in this case.

5 Conclusion and Future Work

In this work we have described an approach to automatically converting DS to PS with significantly improved accuracy over previous efforts, and comparable results to that of using a phrase structure parser guided by the dependency structure.

Following the motivation discussed in Section 1, the next step is straightforward - to adapt the algorithm to work on conversion from a dependency representation of the Arabic Treebank to the phrase structure representation necessary for the annotation pipeline. Following this, we will then experiment with parsing the Arabic dependency representation, converting to phrase structure, and evaluating the resulting phrase structure representation as usual for parsing evaluation. We will also experiment with dependency parsing for the PTB dependency representation discussed in this paper. Habash and Roth (2009) discuss an already-existing dependency representation of parts of the ATB and it will be interesting to compare the conversion accuracy using the different dependency representations, although we expect that there will not be any major differences in the representations.

One other aspect of future work is to implement the algorithm in Wang and Zong (2010), using our own dependency representation, since this would allow a precise investigation of what the phrase structure parser is contributing as compared to our automatic conversion. We note that this work also experimented with dependency parsing, and then automatically converting the results to PS, a further basis of comparison.

Finally, we would like to stress that while we have used evalb for scoring the converting PS because it is the standard evaluation for PS work, it is a very insufficient standard for this work. As discussed at the end of Section 2, we have not included all the function tags or empty categories in our representation, a significant omission. We would like to expand our dependency representation to allow all the function tags and empty categories to be included in the converted PS. Our plan is to take our analogy to TAG more seriously (e.g., (Joshi and Rambow, 2003)) and use a label akin to adjunction to encode leftward (non-projective) movement in the tree, also using an appropriate dependency parser as well (Shen and Joshi, 2008).

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency, GALE Program Grant No. HR0011-06-1-0003. The views, opinions and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense. We would also like to thank Mohamed Maamouri, Colin Warner, Aravind Joshi, and Mitch Marcus for valuable conversations and feedback.

References

Srinivas Bangalore and Aravind K. Joshi, editors. 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. MIT Press.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II-style Penn Treebank project. Technical Report MS-CIS-95-06, University of Pennsylvania.

David Chiang. 2003. Statistical parsing with an automatically extracted Tree Adjoining Grammar. In *Data Oriented Parsing*. CSLI.

Nizar Habash and Ryan Roth. 2009. CATiB: The Columbia Arabic Treebank. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 221–224, Suntec, Singapore, August. Association for Computational Linguistics.

Johan Hall and Joakim Nivre. 2008. A dependency-driven parser for German dependency and constituency representations. In *Proceedings of the Workshop on Parsing German*, pages 47–54, Columbus, Ohio, June. Association for Computational Linguistics.

Joakim Nivre Johan Hall and Jens Nilsson. 2007. Hybrid constituency-dependency parser for Swedish. In *Proceedings of NODALIDA*, Tartu, Estonia.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of NODALIDA*, Tartu, Estonia.

Aravind Joshi and Owen Rambow. 2003. A formalism for dependency grammar based on Tree Adjoining Grammar. In *Proceedings of the Conference on Meaning-Text Theory*, Paris, France.

Mohamed Maamouri and Ann Bies. 2004. Developing an arabic treebank: Methods, guidelines, procedures, and tools. In Ali Farghaly and Karine Megerdoomian, editors, *COLING 2004 Computational Approaches to Arabic Script-based Languages*, pages 2–9, Geneva, Switzerland, August 28th. COLING.

Mohamed Maamouri, Ann Bies, and Seth Kulick. 2011. Upgrading and enhancing the Penn Arabic Treebank. In Joseph Olive, Caitlin Christianson, and John McCary, editors, *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation*. Springer.

Christopher Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing, 12th International Conference, CICLing 2011, Proceedings, Part I. Lecture Notes in Computer Science 6608*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19:313–330.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of HLT*.

Satoshi Sekine and Michael Collins. 2008. Evalb. http://nlp.cs.nyu.edu/evalb/.

Libin Shen and Aravind Joshi. 2008. LTAG dependency parsing with bidirectional incremental construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii, October. Association for Computational Linguistics.

Zhiguo Wang and Chengqing Zong. 2010. Phrase structure parsing with dependency structure. In *COLING 2010: Posters*, pages 1292–1300, Beijing, China, August.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *HLT-2001*.

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2009. Towards a multi-representational treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*.