Intra-Chunk Dependency Annotation: Expanding Hindi Inter-Chunk Annotated Treebank

Prudhvi Kosaraju
Intl. Institute of Info. Technology
Hyderabad, India
prudhvi.kosaraju@research.iiit.ac.in

Bharat Ram Ambati
Intl. Institute of Info. Technology
Hyderabad, India
ambati@research.iiit.ac.in

Samar Husain
Univ. of Potsdam
Potsdam, Germany
husain@uni-potsdam.de

Dipti Misra Sharma
Intl. Institute of Info. Technology
Hyderabad,India
dipti@iiit.ac.in

Rajeev Sangal
Intl. Institute of Info. Technology
Hyderabad, India
sangal@mail.iiit.ac.in

Abstract

We present two approaches (rule-based and statistical) for automatically annotating intra-chunk dependencies in Hindi. The intra-chunk dependencies are added to the dependency trees for Hindi which are already annotated with inter-chunk dependencies. Thus, the intra-chunk annotator finally provides a fully parsed dependency tree for a Hindi sentence. In this paper, we first describe the guidelines for marking intra-chunk dependency relations. Although the guidelines are for Hindi, they can easily be extended to other Indian languages. These guidelines are used for framing the rules in the rule-based approach. For the statistical approach, we use MaltParser, a data driven parser. A part of the ICON 2010 tools contest data for Hindi is used for training and testing the MaltParser. The same set is used for testing the rule-based approach.

1 Introduction

Treebanks are corpora in which each sentence pairs with a parse tree. These are linguistic resources in which the morphological, syntactic and lexical information for each sentence has been explicitly marked. Some notable efforts in this direction are the Penn Tree Bank (Marcus et al., 1993) for English and the Prague Dependency Bank (Hajicova, 1998) for Czech. Lack of such treebanks has been a major bottleneck in various efforts in advance research and development of NLP tools and applications for Indian languages.

Treebanks can be created manually or semi-automatically. Manual creation of treebank is a costly task both in terms of money and time. The annotators follow a set of prescribed guidelines for the annotation task. Semi-automatic creation of treebank involves first running of tools/parsers and then manual correction of errors. An accurate annotating parser/tool saves cost and time for both the annotation as well as the validation task.

A multi-layered Hindi treebank is in the process of being created (Bhatt et al., 2009). Dependency treebank forms the first layer in this annotation. To save annotation effort, manual annotation of the dependency relations for Hindi dependency treebank is carried at the inter-chunk level. The intra-chunk relations are marked automatically. The focus of this paper is the task of automatically marking intra-chunk relations. We present both a rule-based and a statistical approach for this expansion process. We call this process 'expansion' since the intra-chunk dependencies are made explicit by removing the chunk

Proceedings of the 6th Linguistic Annotation Workshop, pages 49–56,
Jeju, Republic of Korea, 12-13 July 2012.

encapsulation; one could visualize this as expanding the chunk into sub-trees. The rest of the paper is organized as follows. Sections 2 & 3 give an overview of Hindi treebank and the steps involved in its development. Section 4 describes the guidelines for annotating intra-chunk dependencies. Section 5 shows our approach to building an automatic intra-chunk annotator. Section 6 talks about issues with a couple of dependency relations and how these are handled by the automatic annotator. We conclude in section 7 and present future work in Section 8.

2 Hindi Dependency Treebank

A multi-layered and multi-representational Treebank for Hindi (Bhatt et al., 2009; Xia et al., 2009) is currently being developed. The treebank will have dependency relations, verb-arguments (PropBank, Palmer et al., 2005) and phrase structure (PS) representations. The dependency treebank contains information encoded at the morpho-syntactic (morphological, part-of-speech and chunk information) and syntactico-semantic (dependency) levels The manual annotation of the dependency treebank entails the annotation of part of speech (POS) tag, morphological information for each word, identification of chunk boundary (and chunk tag) and marking inter-chunk dependency relation between word pairs.

The intra-chunk dependencies are left unannotated. The decision to leave intra-chunk relations unmarked is based on the understanding that their identification is quite deterministic and can be automatically annotated with high degree of accuracy. The notion of chunk is, in essence, used as a device for modularity in the process of annotation. The relations among the words in a chunk are not marked in the initial phase of annotation and hence allow us to ignore local details while building the sentence level dependency tree. An example of inter-chunk dependency annotation is given in Figure 1 below. Note how the two chunks (the noun chunk, NP and the verb chunk, VGF) are related to each other using the attribute 'drel' (dependency relation), also note that the relations between the chunk-internal words (e.g. नीली and किताब in the NP chunk) are left unspecified. The annotation is represented in SSF[1]

Sentence1: नीली किताब गिर गई

niilii kitaab gir gaii

'blue' 'book' 'fall' 'go-perf'

The blue book fell down

```
1     ((      NP     <name='NP' drel='k1:VGF'>
1.1   niilii  JJ     <name='niilii'>
1.2   kitaab  NN     <name='kitaab'>
      ))
2     ((      VGF    <name='VGF'>
2.1   gir     VM     <name='gir'>
2.2   gaii    VAUX   <name='gaii'>
      ))
```

Figure 1: SSF representation

Figure 2 shows the schematic dependency tree for sentence 1.

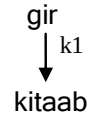


Figure 2: Inter-chunk dependency tree of sentence1

The inter-chunk dependency annotation is done following the dependency guidelines in Bharati et al., (2009) that uses a dependency framework inspired by Panini's grammar of Sanskrit (see, Begum et al., 2008 for more details). Subsequent to inter-chunk dependency annotation, intra-chunk annotation is done automatically following the guidelines described in this paper.

The final treebank for Hindi would have other layers annotation such as Propbank and Phrase structure. The conversion to phrase structure depends on the expanded version of the treebank (i.e. trees with inter-chunk, as well as, intra-chunk relations marked).Hence, it is important to have high quality complete dependency structure for each sentence, and since inter-chunk annotation is manual, this implies that the process of automatic expansion (i.e. the task of making intra-chunk relations explicit) should be very accurate.

[1] SSF: Shakti Standard Format http:// web2py.iiit.ac.in/ publications/default/download/techreport.pdf.c08a8d0a-50ed-4837-8ff0-93d099efbccb.pdf

```
1    niilii    JJ     <fs drel='nmod__adj:kitaab' chunkType='child:NP' name='niilii '>
2    kitaab    NN     <fs drel='k1:gir' name='kitaab' chunkId='NP' chunkType='head:NP'>
3    gir       VM     <fs name='gir' chunkId='VGF' chunkType='head:VGF'>
4    gaii      VAUX   <fs drel='lwg__aux:gir' name='gaii' chunkType='child:VGF'>
```

Figure 3: SSF representation of complete dependency tree

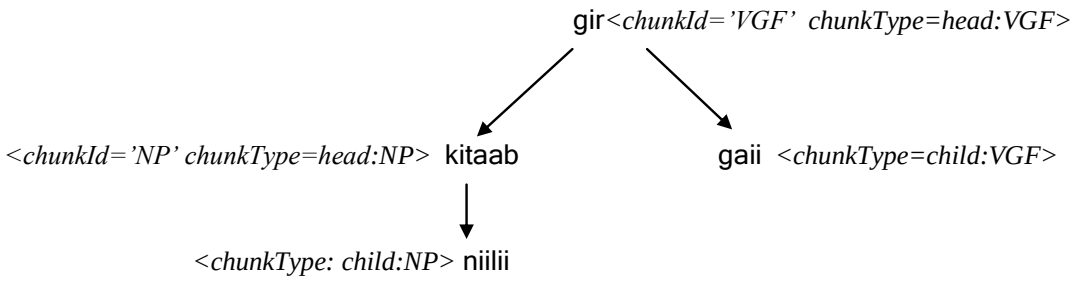


Figure 4: Complete dependency tree of sentence 1

3 Intra-Chunk Annotation

Showing intra-chunk relations and thereby a fully parsed dependency tree implies chunk removal from the inter-chunk dependency annotation. Once the intra-chunk dependencies are made explicit, every sentential token becomes part of the dependency tree. However, it can be useful to retain the chunk information which has been manually validated for inter-chunk dependency annotation. Indeed, previous parsing experiments for Hindi during the ICON2010 tools contest (Husain et al., 2010) have shown that this information consistently improves performance. Thus, during the process of expansion, we introduce two attribute-value pairs for this purpose. This way we maintain chunk information after making the intra-chunk relations explicit. This makes it possible for the users of the treebank to select the chunk head and ignore the intra-chunk information if so desired. Alternatively, it is also possible to access the complete dependency tree.

In Figure 1, the dependency relations are marked between chunk heads, i.e. 'kitaab' is seen related to 'gir' with a 'k1' relation. 'niilii' and 'gaii', on the other hand, are not shown related to any other word. Also note that the chunk boundaries are shown using brackets. Once we show all the tokens as part of the dependency tree, this information goes in the feature structure of individual nodes. This can be seen in figure 3.

The attribute, 'chunkId' and 'chunkType' substitute the bracketing, as well as show the chunk members in the role of head and child. The head node has 'chunkId' that gives it a unique chunk name; note that this is same as the value of 'name' for the original chunk. When multiple chunks with same name occur in a sentence, we append a number along with the name. For example, if there are multiple NP's then the chunk ids will be NP, NP2 and NP3 etc. In addition, all the chunk members have 'chunkType' that gives their membership type. In the example (figure 3), the adjective 'nIlI' modifies the head noun 'kiwAba' with 'nmod__adj' relation. The chunk membership is also shown for both these tokens, nIlI is the 'child of the chunk with chunkId=NP' shown by chunkType. kiwAba on the other hand is the 'head of the chunk with chunkId=NP', it has both chunkType and chunkId.

4 Intra-Chunk Dependency Guidelines

Intra-chunk labels are used when the dependencies within a chunk are made explicit. There are a total of 12 major intra-chunk tags. The tags are of three types: (a) normal dependencies, eg. *nmod__adj, jjmod__intf,* etc., (b) local word group dependencies(lwg), eg. *lwg__psp, lwg__vaux, etc.*, and (c) linking lwg dependencies, eg. *lwg_cont*. Local word dependencies themselves can be

broadly classified into two types, one that handles post-positions and auxiliary verbs and the other that handles negations, particles, etc. Following guidelines are used to annotate the intra-chunk dependencies.

1. ***nmod__adj:*** Various types of adjectival modifications are shown using this label. An adjective modifying a head noun is one such instance. The label also incorporates various other modifications such as a demonstrative or a quantifier modifying a noun.

Chunk: नीली किताब

NP ((*niilii*_JJ *kitaab*_NN))
'blue ' 'book'

niilii
↓ *nmod__adj*
kitaab

In the above example NP is the chunk with words 'niilii' (blue) and 'kitaab' (book) with POS tags JJ and NN respectively.

2. ***lwg__psp:*** This relation is used to attach post-positions/auxiliaries associated with the noun or a verb. 'lwg' in the label name stands for local word grouping and associates all the postpositions with the head noun. These relations are distinct from normal dependency relations as they are more morphological in nature.

Chunk: अभिषेक ने

NP((*abhishek*_NNP *ne*_PSP))
'abhishek' 'ERG'

abhishek
↓ *lwg__psp*
ne

3. ***lwg__neg:*** This relation is used for negative particles. Negative particles are normally grouped with a noun/verb. Like postpositions or auxiliaries these are also classified as 'lwg'.

Chunk: नहीं आयेगा

VGF((nahim_NEG aayegaa_VM))
'Never' 'Come'

nahim
↓ *lwg__neg*
aayega

4. ***lwg__vaux:*** This relation is used when an auxiliary verb modifies the main verb.

Chunk: हो गया

VGF((ho_VM gayaa_VAUX))
'be' 'go-perf'

ho
↓ *lwg__vaux*
gayaa

5. ***jjmod_intf :*** This relation is used when an adjectival intensifier modifies an adjective.

Chunk: बहुत तेज़ जानवर

NP((bahut_INTF tez_JJ jaanvar_NN))
'very' 'fast' 'animal'

bahut
↓ *nmod__adj*
tez
↓ *jjmod__intf*
jaanvar

6. ***pof__redup:*** This relation is used when there is reduplication inside a chunk. The POS tag will in almost all the cases help us identify such instances. We see this in the example below.

Chunk: धीरे धीरे

RBP((dhiire_RB dhiire_RDP))
'slowly' 'slowly'

dhiire
↓ *pof__redup*
dhiire

7. ***pof__cn:*** This relation is used for relating the components within a compound noun. Like 'pof__redup' identifying such cases will be straight-forward. The POS will provide us with the relevant information

Chunk: रामबच्चन यादव

NP((raamabachhan_NNPC yaadav_NNP))
'rambachhan' 'yadav'

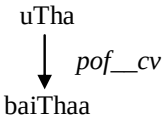
raamabachhan
↓ *pof__cn*
yaadav

8. ***pof__cv :*** This relation is used for compound verbs. Like the previous 'pof' labels, POS

information will be sufficient to identify this relation.

Chunk: उठ बैठा

VGF((uTha_VMC baiThaa_VM))
'rise' 'sit-perf'

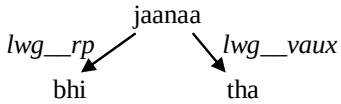


9. ***rsym:*** Punctuation marks and symbols like '-' should be attached to the head of the chunk with relation rsym.

10. ***lwg__rp:*** This relation is used when a particle modifies some chunk head.

Chunk: जाना भी था

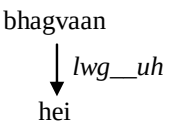
VGF((jaanaa_VM bhi_RP tha_VAUX))
'go-inf' 'also' 'perf'



11. ***lwg__uh:*** This relation is used when interjection modifies other words.

Chunk : हे भगवान

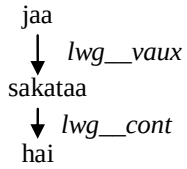
NP((hei_INJ bhagvaan_NN))
'Oh!' 'God'



12. ***lwg__cont:*** We use this label to show that a group of lexical items inside a chunk together perform certain function. In such cases, we do not commit on the dependencies between these elements. We see this with complex post-positions associated with a noun/verb or with the auxiliaries of a verb. 'cont' stands for continue.

Chunk: जा सकता है

VGF((*jaa*_VM *sakataa*_VAUX *hai*_VAUX))
'go' 'can' 'be-pres'



5 Intra-Chunk Dependency Annotator

In this section we discuss our approach to building an intra-chunk dependency annotator/parser for Hindi. We describe three experiments; the first two are rule-based and statistical based, while the third is hybrid in a sense that it adds on a heuristic based post-processing component on top of the statistical technique. We evaluate about approaches in section 5.3 after describing rule-based and statistical approaches in sections 5.1 and 5.2 respectively.

5.1 Rule-Based Dependency Annotator

The rule-based approach identifies the modifier-modified (parent–child) relationship inside a chunk with the help of the rules provided in a rule template. The inter-chunk dependency annotated data is run through a head computation module (a rule-based tool), which marks the head of each chunk. After getting the heads for each chunk, we get the intra-chunk relations using a rule-base that has been manually created. The design of the rule template allows capturing all the information in a SSF representation. The rule template is a 5-columned table with each row representing a rule. Table1 shows a sample rule written using the rule template. The five columns are

1. Chunk Name: Specifies the name of the chunk for which this expansion rule can be applied.

2. Parent Constraints: Lexical item which satisfies these constraints will be identified as the parent. Constraints are designed capturing POS, chunk, word and morphological features. In Table1 the constraint on the parent is specified using its POS category (NN: common noun).

3. Child Constraints: Lexical item satisfying these constraints becomes the child. Constraints are designed similar to the parent constraints. In Table 1 the constraint on the child is specified using its POS category (JJ:adjective).

Chunk Name	Parent Constraints	Child Constraints	Contextual Constraints	Dep. Relation
NP	*POS* == NN	*POS* == JJ	*posn*(parent) > *posn*(child);	nmod__adj

Table 1: Sample rule

4. Contextual Constraints: Lexical items satisfying constraints 1, 2 &3 become parent and child in a chunk. One can access the previous and next words of parent and child by applying arithmetic on *posn* attribute. Information about the lexical item can be accessed by applying attributes like POS (for part of speech tag), CAT (category), and LEMMA (for root form of lexical item).

Here an example of a contextual constraint taken from Table1:

***posn*(parent) > *posn*(child)**

Parent and child constraint look at the properties of word but there are cases where the constraint needs to be formed beyond word level information. These constraints involve capturing of word order information. In such cases we use the operator '>'. It can be used only when '*posn*' attribute is used. Here the constraint means that this rule is applicable only when child occurs before parent inside the chunk.

One can also specify constraints in form of:

POS__posn(parent) - 1 == NN

Here the Part of Speech of word preceding parent is accessed and compared with NN. ***posn(parent) – 1*** retrieves the position of preceding word of parent and ***POS__*** on this position gives us the Part of Speech tag of that lexical item.

5. Dependency Relation: If all the constraints are satisfied, then the dependency relation from this column is marked on the parent-child arc.

5.2 Sub-tree Parsing using MaltParser

We use MaltParser (Nivre et al., 2007) as an alternative method to identify the intra-chunk relations. It is well known in the literature that transition-based dependency parsing techniques (e.g. Nivre, 2003) work best for marking short distance dependencies in a sentence. As must be clear by now, intra-chunk relations are in fact short distance dependencies; and we basically use MaltParser to predict the internal structure of a chunk. So instead of using it to parse a sentence, we parse individual chunks. Each chunk is treated as a sub-tree. The training data contains sub-trees with intra-chunk relations marked between chunk-internal nodes, the head of the chunk becomes the root node of the sub-tree. The MaltParser is trained on these sub-trees and a model is created. We run the test data on this model for marking intra-chunk dependencies among the sub-trees and then post-process them to obtain complete dependency tree for the data.

5.3 Results

In this section we evaluate the three approaches that were explored to build the automatic intra-chunk annotator. A total of 320 sentences extracted from the ICON2010 tools contest data for Hindi (Husain et al., 2010) have been manually annotated for intra-chunk relations. Table 2 shows the statistics for this gold data that has been used for evaluation (and training).

Data	Number of Sentences
Training	192
Development	64
Testing	64

Table 2: Gold data

Rule-Based Approach: As discussed in section 5.1, the rule-based approach marks dependency relation mainly by using POS patterns in a chunk. Table 3 shows the result when evaluated for the test data.

LAS	97.89
UAS	98.50
LS	98.38

Table 3: Parsing accuracies[2] obtained using rule-based tool

Statistical/MaltParser-based approach: Table 2 shows the division of data into training, development and test. The experimentation procedure is similar to the one used in Kosaraju et al., (2010). We prepared a list of features with the aim of getting a better parse. A simple forward selector is used to prune the list and prepare the best feature template. The selector's task is to include the feature into feature template if this

[2] Parsing Accuracies- LAS: labeled attachment score, UAS: Unlabeled attachment score, LS: label score.

template improves the LAS score over the previous template. These feature optimization experiments were conducted over 5-fold cross-validation of the combined training and development data. The best feature template was used to get the final accuracies for the test data. Table 4 shows results on the basic template, template capturing POS patterns and best template that included POS, lemma and other information present in the SSF data.

	LAS	**UAS**	**LS**
Base line	95.70	97.07	96.80
POS template	96.80	97.62	97.80
Best template	**97.35**	**98.26**	**97.90**

Table 4: Parsing accuracies using MaltParser

The POS-based template scores can be compared with the results obtained from the rule-based scores (Table 3) since the rules are formed using POS patterns.

We see that both rule-based and statistical approach give very high accuracies on the test data. These results validate our initial intuition that identification of intra-chunk relations is quite deterministic. These results also support our annotation design choice of leaving the annotation of intra-chunk relations out of the initial manual phase. Table 5 shows percentage error contribution of some major tags to total Error of their respective systems. Table 6 shows precision (P) and recall (R) of some major tags.

Depn. Relation	**Rule-based appraoch**	**Statistical appraoch**
pof__cn	28.33	26.7
nmod__adj	13.3	13.3
lwg__rp	6.6	0
rsym	16.7	20.0

Table 5: Percentage Contribution of error by each tag to the total error of the system

Hybrid approach: Table 5 & 6 shows error analysis of both approaches. For some tags like *nmod__adj* we see the rule-based appraoch shows better results. Therefore we decided to include rules as a post-processing step in the statistical approach.

Depn. Relation	**Rule-based**		**Statistical**	
	P	R	P	R
pof__cn	95.63	**94.50**	91.07	92.73
nmod__adj	96.33	**98.33**	95.28	98.06
lwg__rp	97.62	95.35	100	**100**
rsym	96.71	**97.63**	92.41	96.05

Table 6: Error analysis of both methods

We made the statistical approach hybrid by post-processing the output of the MaltParser. This involves correction of some dependency relations based on heuristics framed from the rules of the rule-based tool. Heuristics are formed for those dependency relations that have higher recall in the rule-based approach compared to the statistical approach. The modification resulted in improvement in parsing accuracies. This can be seen in Table 7.

Approach	**LAS**	**UAS**	**LS**
Rule-based	97.89	98.50	98.38
Statistical	97.35	98.26	97.90
Hybrid	**98.17**	**98.81**	**98.63**

Table 7: Parsing accuracies

6 Special Cases

The neat division between the task of inter-chunk parsing and intra-chunk parsing is based on the following assumption: 'Chunks are self contained units. Intra-chunk dependencies are chunk internal and do not span outside a chunk.' However, there are two special cases where this constraint does not hold, i.e. in these two cases a chunk internal element that is not the head of the chunk has a relation with a lexical item outside its chunk and therefore, these two relations have to be handled separately. These are related to punctuation and co-ordination.

1. rsym__eos: The EOS (end-of-sentence) marker occurs in the last chunk of the sentence. It attaches to the head of the sentence (which may lie in the same chunk or another chunk) with this relation.

2. lwg__psp: As noted in section 4, a PSP (postposition) attaches to the head of its chunk with a lwg__psp relation. However, if the right most child of a CCP (conjunction chunk) is a

nominal (NP or VGNN), one needs to attach the PSP of this nominal child to the head of the CCP during expansion. If there are multiple PSP then the first PSP gets lwg__psp and the following gets lwg__cont relation. Take the following example

NP(*raama*_NNP) CCP(*aur*_CC) NP(*siitaa*_NNP *ne*_PSP)
'ram' 'and' 'sita'
'ERG'

In this case the PSP connects to the CC with the relation lwg__psp. The subtree after expansion is shown in figure 6.

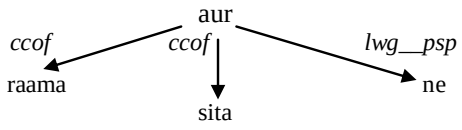


Figure 6: Expanded sub-tree with PSP connected with CC.

7 Conclusion

In this paper we described annotation guidelines for marking intra-chunk dependency relations. We then went on to show that these relations can be automatically identified with high accuracy. This was illustrated using (1) a rule-based approach that mainly used intra-chunk POS patterns, and (2) a statistical approach using MaltParser. We also showed that these two systems can be combined together to achieve even higher accuracy.

From the report of error analysis, it is been shown that there are certain relations that are not being marked successfully. This is good news because then one can make very targeted manual corrections after the automatic tool is run.

Acknowledgments

We would like to thank Karthik Gali for his initial efforts in building the annotator. The work reported in this paper is supported by the NSF grant (Award Number: CNS 0751202; CFDA Number: 47.070).

References

A. Bharati, D. M. Sharma S. Husain, L. Bai, R. Begam and R. Sangal. 2009. AnnCorra: TreeBanks for Indian Languages, Guidelines for Annotating Hindi TreeBank (version–2.0). http://ltrc.iiit.ac.in/MachineTrans/research/tb/DSguidelines/DS-guidelines-ver2-28-05-09.pdf

E. Hajicova. 1998. Prague Dependency Treebank: From Analytic to Tectogrammatical Annotation. *In Proc. TSD'98.*

F. Xia, O. Rambow, R. Bhatt, M. Palmer, and D. M. Sharma. 2009. Towards a Multi-Representational Treebank. *In Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories (TLT 2009), Groningen, Netherlands.*

J. Nivre, An Efficient Algorithm for Projective Dependency Parsing. *In Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03), Nancy, France, 23-25 April 2003, pp. 149-160.*

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov and E Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering, 13(2), 95-135.*

M. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English : The Penn Treebank. *Computational Linguistics 1993.*

P. Kosaraju, S. R. Kesidi, V. B. R. Ainavolu and P. Kukkadapu. 2010. Experiments on Indian Language Dependency Parsing. *In Proceedings of the ICON10 NLP Tools Contest: Indian Language Dependency Parsing.*

R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai, and R. Sangal. 2008. Dependency annotation scheme for Indian languages. *In Proceedings of IJCNLP-2008.*

R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma and F. Xia. 2009. MultiRepresentational and Multi-Layered Treebank for Hindi/Urdu. *In Proc. of the Third Linguistic Annotation Workshop at 47th ACL and 4th IJCNLP.*