Empty Argument Insertion in the Hindi PropBank

Ashwini Vaidya, Jinho D. Choi, Martha Palmer, Bhuvana Narasimhan

Institute of Cognitive Science
University of Colorado at Boulder
{vaidyaa,choijd,mpalmer,narasimb}@colorado.edu

Abstract

This paper examines both linguistic behavior and practical implications of empty argument insertion in the Hindi PropBank. The Hindi PropBank is annotated on the Hindi Dependency Treebank, which contains some empty categories but rarely the empty arguments of verbs. In this paper, we analyze four kinds of empty arguments, ⋆PRO⋆, ⋆REL⋆, ⋆GAP⋆, ⋆pro⋆, and suggest effective ways of annotating these arguments. Empty arguments such as ⋆PRO⋆ and ⋆REL⋆ can be inserted deterministically; we present linguistically motivated rules that automatically insert these arguments with high accuracy. On the other hand, it is difficult to find deterministic rules to insert ⋆GAP⋆ and ⋆pro⋆; for these arguments, we introduce a new annotation scheme that concurrently handles both semantic role labeling and empty category insertion, producing fast and high quality annotation. In addition, we present algorithms for finding antecedents of ⋆REL⋆ and ⋆PRO⋆, and discuss why finding antecedents for some types of ⋆PRO⋆ is difficult.

Keywords: Hindi, PropBank, dependency structure, semantic role labeling, empty category insertion

1. Introduction

The Hindi PropBank is part of the multi-representational and multi-layered resource creation project for the Hindi-Urdu language (Bhatt et al., 2009). It is a corpus in which arguments of verbal predicates (including complex predicates such as light verbs) are annotated with their semantic roles. Two aspects distinguish the Hindi PropBank from other PropBanks. First, the annotation is done on top the Hindi Dependency Treebank (Begum et al., 2008), whereas PropBanks in most other languages are annotated on constituent-based Treebanks (Palmer et al., 2005; Xue, 2008; Zaghouani et al., 2010). One clear advantage of using dependency structure for annotating predicate argument structure is that it is possible to find mappings between syntactic and semantic dependencies and use these mappings to automatically identify semantic arguments of verb predicates from their syntactic dependents (Vaidya et al., 2011).

The other unique aspect of the Hindi PropBank is motivated by dropped arguments. Hindi is a language that regularly drops required arguments that are recoverable from prior discourse (e.g., "(vo) kitaab paRegaa"; "(*He*) *will read the book*"). The Hindi Dependency Treebank does not represent such dropped arguments although it includes some other empty categories such as empty nouns (e.g., ellipsis), empty verbs (e.g., gapping, sluicing), empty conjunctions, etc. (Bhatia et al., 2010). Thus, to give complete representations of predicate argument structures including dropped arguments, the annotation task for the Hindi PropBank consists of semantic role labeling as well as empty argument insertion. The empty argument insertion not only captures the semantic information contained in elided arguments (Section 2.), but also aids in the automatic conversion from dependency structure to phrase structure (Xia et al., 2009).

Although inserting empty arguments gives richer semantic representations, it presents some practical difficulties for annotation. Empty argument insertion usually requires intensive syntactic analysis with expert linguistic knowledge that is not necessarily required for semantic role labeling. Thus, it takes a lot of time to train annotators for accomplishing both tasks, which slows down the whole annotation process. Our first approach was to carry out both tasks in a pipeline, inserting all empty arguments first then annotating semantic roles given the empty arguments. We found this pipeline approach to be impractical: manual insertion of empty arguments was so time consuming that it often became a bottleneck to semantic role labeling. Furthermore, inserting empty arguments without considering their semantic roles caused over-generation of these arguments.

Our second approach was to insert all empty arguments automatically using linguistically motivated rules. This approach gave very good results for certain kinds of empty arguments (e.g., empty relative pronouns), but not all (Section 3.). For the empty arguments that our rules did not handle well, we designed a new annotation scheme that concurrently handled both semantic role labeling and empty argument insertion (henceforth, we refer to this as a "joint annotation" of these two tasks). This required a bit more training for annotators but resulted in a much simpler and more efficient annotation process, producing higher quality annotation (Section 4.). The joint annotation was carried out using a single annotation tool, Jubilee (Choi et al., 2010), which provided verb frame information; this frame information helped annotators to decide whether or not an empty argument was required for a particular verb.

Our next challenge was to facilitate this annotation process by applying automatic antecedent resolution. We came up with algorithms for finding antecedents of automatically inserted empty arguments, which showed high accuracy for certain cases but had difficulties with others (Section 5.). By applying automatic empty category insertion and automatic antecedent resolution, we gained more consistent annotation while still taking less annotation time.

2. Descriptions of empty arguments

The Hindi PropBank extends the structural analysis of the Hindi Dependency Treebank by adding arguments missing

from predicate argument structure. Four kinds of empty arguments have been added. Note that all the empty arguments inserted in the Hindi PropBank are core arguments, i.e., the subjects, objects or in some cases indirect objects of verbs. Thus, they are always annotated as ARG0, ARG1, or ARG2 in PropBank, which correspond to the prototypical agent, patient, or recipient, respectively (Vaidya et al., 2011). The following examples show descriptions of each empty argument (for more details, Bhatia et al. (2010)).

- ⋆PRO⋆: The subject of a non-finite clause (controlled by either its subject or object) which is obligatorily absent. The non-finite clause can be either a complement or adjunct clause. Figure 1 shows an example of subject control in a complement clause. ⋆PRO⋆ is inserted as an argument of the verb *read* and co-referenced to *Mohan*.

मोहन_ने$_i$	*PRO*$_i$	किताब	पढ़नी	चाही
Mohan_ERG	(he)	book	read_INF	want_PERF

Figure 1: An example of ⋆PRO⋆: "*Mohan wanted* ⋆PRO⋆ *to read the book*" in Hindi.

- ⋆REL⋆: The subject or object of a participial relative clause that is obligatorily absent and refers to the modified noun. In Figure 2, ⋆REL⋆ is inserted as an argument of the verb *run*.

REL	भागता_हुआ	लड़का	सेब	खा_रहा_है
(who)	run_INF_be	boy	apple	eat_CONT_be

Figure 2: An example of ⋆REL⋆: "*The* ⋆REL⋆ *running boy is eating an apple*" in Hindi.

- ⋆GAP⋆: The subject, object, or indirect object that is elided, usually in a co-ordination structure. In Figure 3, ⋆GAP⋆ is inserted as an argument of the verb *sleep* and co-referenced to *Mohan*.

मोहन_ने$_i$	किताब	पढ़ी	और	*GAP*$_i$	सो_गया
Mohan_ERG	book	read_PERF	and	(he)	sleep_go_PERF

Figure 3: An example of ⋆GAP⋆: "*Mohan read the book and* ⋆GAP⋆ *slept*" in Hindi.

- ⋆pro⋆: The subject, object, or indirect object elided for discourse-pragmatic reasons. In Figure 4, the subject of the verb *buy* is elided because it can be recoverable from prior discourse or context.

इसके	बाद	(*pro*)	गाडी	भी	खरीदी
This_GEN	after	(he)	car	also	buy_PST

Figure 4: An example of ⋆pro⋆: "*After this,* ⋆pro⋆ *also bought a car*" in Hindi.

3. Linguistically motivated rules

Table 1 shows an algorithm using linguistically motivated rules for inserting ⋆PRO⋆ and ⋆REL⋆. VGNF, VGINF, and VGNN represent POS tags of non-finite verbs, infinitival verbs, and gerunds, respectively (Bharati et al., 2006). k1 represents a dependency relation for the "locus of activity", and nmod__⋆inv represents any dependency relation for noun modifiers of inverse type (for more details, see Sharma et al. (2009)). For each verb, ⋆PRO⋆ and ⋆REL⋆ are inserted as children of the verb. In line 5, a pre-defined dictionary is used to identify intransitive verbs; this dictionary is generated by analyzing the general behavior of verb predicates. The dictionary also contains information about the type of argument for intransitive verbs, which is used to check the condition in line 6.

Input: t, where t is a dependency tree.
Post: ⋆PRO⋆ and ⋆REL⋆are inserted in t.
begin
1: **foreach** v **in** t: # v is a verb predicate
2: **if** v.pos = VGNF | VGINF | VGNN:
3: **if** v.deprel = nmod__⋆inv:
4: v.addChild(⋆REL⋆);
5: **elif** v is an *intransitive* verb:
6: **if** v's required dependent doesn't exist:
7: v.addChild(⋆PRO⋆);
8: **else**: # v is a (*di*)*transitive* verb
9: **if not** v.existChild(k1) :
10: v.addChild(⋆PRO⋆);
end

Table 1: An algorithm for inserting ⋆PRO⋆ and ⋆REL⋆.

Table 2 shows how accurately this algorithm inserts ⋆PRO⋆ and ⋆REL⋆ in our corpus. 37K and 47K are two separate batches from the same corpus containing about 37K and 47K word-tokens, respectively. We evaluate these batches separately because dependency trees in 37K are checked very thoroughly whereas trees in 47K are not checked as thoroughly; in other words, dependency trees in 47K contain more annotation errors than ones in 37K, which affect the evaluation of automatic empty argument insertion. Given the rich set of dependency relations provided by the Hindi Dependency Treebank, we achieve 100% precision for ⋆REL⋆ in both batches. Furthermore, from our error analysis, we verify that most recall errors for ⋆REL⋆ are caused by inconsistent annotations in the Treebank. We also obtain good results for ⋆PRO⋆; especially for recall. It may be possible to improve the precision of ⋆PRO⋆ by refining our dictionary and discovering more rules, which we will explore in future work.

	Type	Precision	Recall	F1-score
37K	⋆PRO⋆	87.79	93.69	90.64
	⋆REL⋆	100.00	94.32	97.08
47K	⋆PRO⋆	85.02	91.91	88.33
	⋆REL⋆	100.00	89.92	94.69

Table 2: Insertion accuracies of ⋆PRO⋆ and ⋆REL⋆ (in %).

Table 3 shows the distributions of empty arguments in both 37K and 47K (37K consists of 2,710 verb predicates and 47K consists of 3,055 verb predicates). *PRO* and *REL* are the most frequently occurring empty arguments whereas *GAP* and *pro* occur less frequently.

Corpus	*PRO*	*REL*	*GAP*	*pro*
37K	490	176	33	96
47K	553	238	24	76

Table 3: Numbers of empty arguments in our corpus.

4. Joint annotation of semantic role labeling and empty argument insertion

For *GAP* and *pro*, we devised rules that made use of the valency requirements of a given verb and cues like coordination structure for automatic insertion. However, our approach resulted in over-generation of these arguments with low precision. Inserting elided *pro* required the identification of the relevant sense of the predicate, which our rules could not access. In the case of *GAP*, the coordination structure provided an important cue, but the identification of a shared argument context, where the *GAP*had an antecedent in the coordinated clause, was difficult.

Since automatic insertion was not successful for these cases, we decided to annotate them manually using a new annotation scheme, called a joint annotation of semantic role labeling and empty argument insertion, that concurrently handled these two tasks. For the case of shared arguments such as *GAP*, we carried out joint annotation on their antecedents. In Figure 5, the presence of *GAP* for the verb *sleep* is indicated on its antecedent, *Mohan*, such that the dependency relation GAP-ARG0 is assigned to *Mohan* from *sleep*. The subsequent post-processing step automatically inserts *GAP* as a dependent of *sleep*, and assigns the semantic role ARG0 to it. The co-reference between *Mohan* and *GAP* is also deterministically retrieved during this post-processing step.

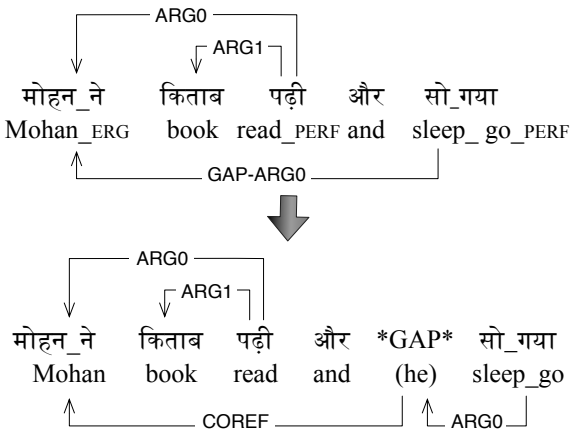

Figure 5: A joint annotation of *GAP*: "*Mohan read the book and* *GAP* *slept*" in Hindi (from Figure 3).

In contrast to *GAP*, the case of *pro* does not have an antecedent in the clause. In such a case, joint annotation is done on the verb predicate. In Figure 6, the verb *buy* has a missing subject so it is indicated on the verb itself with the label pro-ARG0. The subsequent post-processing step automatically inserts *pro* as a dependent of *buy* and assigns the semanic role ARG0 to it. The positions of empty arguments are determined by their semantic roles: ARG0 to be the first, ARG1 to be the last, and ARG2 to be the second last child of the verb.

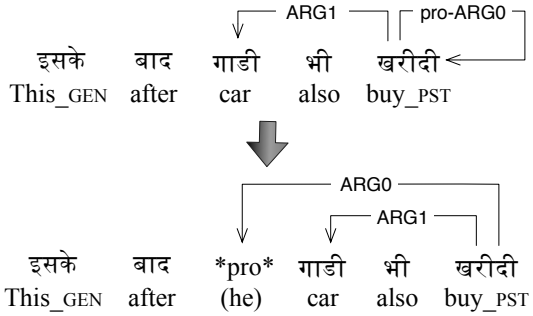

Figure 6: A joint annotation of *pro*: "*After this,* *pro* *also bought a car*" in Hindi (from Figure 4).

This joint annotation scheme can also be used for *PRO* and *REL* in cases where the algorithm in Table 1 cannot identify them correctly. Several advantages have been obtained from this approach. First, considering semantic roles of these empty arguments during the insertion improves precision, producing much higher quality annotation. Second, both semantic role labeling and empty argument insertion can be done using one annotation tool, Jubilee (Choi et al., 2010), which makes it easier to maintain different layers of annotations consistently. Third, verb frame information provided by Jubilee helps annotators in deciding whether or not a particular empty argument is required for a given verb predicate, the information that was not available when the insertion was done as a separate pass.

5. Discussion

While our results for *REL* show 100% precision in Table 2, recall for *REL* is not nearly as perfect. Most of these recall errors are caused by label errors in the Treebank where the dependency label nmod (noun modifier) is given instead of nmod__*inv (noun modifier of inverse type) in which case, the rules for *REL* insertion fail. Furthermore, many errors for *PRO* are caused by non-finite verbs with k1 dependents (locus of activity), which are skipped by our insertion rules. These are cases where either the Treebank uses the k1 label erroneously, or the non-finite verb belongs to the class of 'experiencer verbs' whose objects are marked with the k1 label.

In addition to automatic insertion of empty categories, we also carry out antecedent resolution as this allows annotators to quickly locate the shared arguments. Tables 4 and 5 show algorithms for finding antecedents of *REL* and *PRO*. Our experiments show good results for *REL* (an F1-score of 99.30%) but poor results for *PRO* (an F1-score of 54.35%). One of the difficulties with antecedent resolution for *PRO* is the presence of the 'arbitrary' PRO. This is a special subtype of *PRO* which does not have an antecedent (e.g. "*PRO* Drinking milk is

healthy"). In this example, the ⋆PRO⋆ for the verb *Drinking* can refer to any person in general such that it cannot be co-referential with another element in the sentence. At this point, we have not found clear rules to distinguish between the cases of arbitrary PRO and the typical cases of subject/object controlled PRO. Other complications include cases where ⋆PRO⋆ is co-referential with ⋆pro⋆. Such cases are difficult to resolve when no prior annotation of ⋆pro⋆ exists.

Input: t, where t is a dependency tree.
Post: Antecedents of all ⋆REL⋆ in t are found.
begin
1: **foreach** ⋆REL⋆ **in** t:
2: **let** v **be** the verb predicate of ⋆REL⋆
3: **let** h **be** the dependency head of v
4: ⋆REL⋆.antecedent ← h
end

Table 4: An algorithm for finding antecedents of ⋆REL⋆.

In Table 4, given a ⋆REL⋆ and its verb predicate v, the dependency head of v becomes the antecedent of the ⋆REL⋆.

Input: t, where t is a dependency tree.
Post: Antecedents of subject/object controlled ⋆PRO⋆ in t are found.
begin
1: **foreach** ⋆PRO⋆ **in** t:
2: **let** v **be** the verb predicate of ⋆PRO⋆
3: **let** h **be** the nearest dependency ancestor of v whose POS is VGF
4: **if** h exists:
5: d ← null
6: **if** h.existChild(k1): d ← k1 of h
7: **elif** h.existChild(k4a): d ← k4a of h
8: **elif** h.existChild(k4): d ← k4 of h
9: **if** $d \neq$ null and d is prior to v:
10: ⋆PRO⋆.antecedent ← d
end

Table 5: An algorithm for finding antecedents of ⋆PRO⋆.

In Table 5, given a ⋆PRO⋆ and its verb predicate v, the algorithm first finds a finite verb (VGF) that is the nearest dependency ancestor of v, and checks if the finite verb has a k1 (locus of activity), k4a (experiencer), or k4 (beneficiary) dependent. If there is such a dependent d whose word-order is prior to v, the algorithm makes d the antecedent of the ⋆PRO⋆.

6. Conclusion

We present an algorithm for automatic insertion of the two most frequently occurring empty arguments in the Hindi PropBank, ⋆PRO⋆ and ⋆REL⋆, with high accuracy. This algorithm involves analyzing linguistic behavior of these empty arguments as well as exploiting the rich set of dependency relations in the Hindi Dependency Treebank. The automatic insertion of ⋆PRO⋆ and ⋆REL⋆ covers nearly 83% of the total number of empty arguments in the representative corpus. For ⋆GAP⋆ and ⋆pro⋆, we propose a joint annotation scheme that efficiently combines the annotation tasks of semantic role labeling and empty argument insertion. This annotation scheme can also be used for ⋆PRO⋆ and ⋆REL⋆, where the automatic system cannot identify them correctly. The joint annotation scheme has resulted in faster productivity with higher quality annotation compared to our previous pipeline approach.

As a pilot study, we present algorithms for finding antecedents of ⋆REL⋆ and ⋆PRO⋆. Automatic antecedent resolution for ⋆REL⋆ can be done very accurately whereas more work is required to achieve good accuracy for finding antecedents of ⋆PRO⋆. Once we have enough annotation, we will explore the possibility of using machine learning techniques to improve automatic empty argument insertion and antecedent resolution.

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 the 3rd International Joint Conference on Natural Language Processing*, IJCNLP'08.

Akshar Bharati, Rajeev Sangal, Dipti Misra Sharma, and Lakshmi Bai. 2006. AnnCorra : Annotating Corpora Guidelines For POS And Chunk Annotation For Indian Languages. Technical Report TR-LTRC-31, IIIT, Hyderabad.

Archna Bhatia, Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Michael Tepper, Ashwini Vaidya, and Fei Xia. 2010. Empty Categories in a Hindi Treebank. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*, LREC'10, pages 1863–1870.

Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. A Multi-Representational and Multi-Layered Treebank for Hindi/Urdu. In *Proceedings of ACL-IJCNLP Workshop on Linguistic Annotation*, LAW'09, pages 186–189a.

Jinho D. Choi, Claire Bonial, and Martha Palmer. 2010. Propbank Instance Annotation Guidelines Using a Dedicated Editor, Jubilee. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*, LREC'10, pages 1871–1875.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106.

Dipti Misra Sharma, Rajeev Sangal, Lakshmi Bai, and Rafiya Begam. 2009. AnnCorra : TreeBanks for Indian Languages (version - 2.0). Technical report, IIIT, Hyderabad.

Ashwini Vaidya, Jinho D. Choi, Martha Palmer, and Bhuvana Narasimhan. 2011. Analysis of the hindi proposition bank using dependency structure. In *Proceedings of ACL workshop on Linguistic Annotation*, LAW'11, pages 21–29.

Fei Xia, Rajesh Bhatt, Owen Rambow, Martha Palmer, and Dipti Misra Sharma. 2009. Towards a Multi-representational Treebank. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories*, TLT'7, pages 159–170.

Nianwen Xue. 2008. Labeling Chinese predicates with semantic roles. *Computational Linguistics*, 34(2):225–255.

Wajdi Zaghouani, Mona Diab, Aous Mansouri, Sameer Pradhan, and Martha Palmer. 2010. The Revised Arabic PropBank. In *Proceedings of ACL workshop on Linguistic Annotation*, LAW'10, pages 222–226.