Samajh-Boojh: A Reading Comprehension System in Hindi

Shalaka Vaidya*[1], Hiranmai Sri Adibhatla*[2], Radhika Mamidi[3]
Language Technologies Research Centre
Kohli Center on Intelligent Systems
International Institute of Information Technology - Hyderabad
[1]`hello@shalakavaidya.me`
[2]`hiranmai.sri@research.iiit.ac.in`
[3]`radhika.mamidi@iiit.ac.in`

Abstract

This paper presents a novel approach designed to answer questions on a reading comprehension passage. It is an end-to-end system which first focuses on comprehending the given passage wherein it converts unstructured passage into a structured data and later proceeds to answer the questions related to the passage using solely the aforementioned structured data. To the best of our knowledge, the proposed model is first of its kind which accounts for entire process of comprehending the passage and then answering the questions associated with the passage. The comprehension stage converts the passage into a **Discourse Collection** that comprises of the relation shared amongst logical sentences in given passage along with the key characteristics of each sentence. This model has its applications in academic domain and query comprehension in speech systems among others.

1 Introduction

The Samajh-Boojh[1] system which we have built focuses on the basic principles behind utilization of rules in order to capture the semantics of the given passage which is in the Devanagari script. The current trend is towards incorporating machine learning in the question answering models but they come with a downside of requiring huge quantity and variety of training data to achieve decent accuracy. Whereas the proposed model is rule-based and hence eliminates the need for extensive training data while still providing 75% accuracy.

The Samajh-Boojh system answers 11 types of questions (Table 1) using approximately 25 rules. This sheds light on the fact that, with substantially less number of rules a wide range of questions can be answered. It is an extension to Prashnottar model (Sahu et al., 2012) which could handle 4 types of questions using 4 rules. The system can be classified into two parts, the comprehension part and the question answering part, these two parts together ensure that the system behaves similar to the way humans approach the questions which are asked based on reading comprehension passage. The comprehension part of the system converts the given passage whose inherent structure cannot be grasped by the machine to a structured and machine extractable data called Discourse Collection. The Discourse Collection is then sent to the QA system as an input along with the query to obtain the relevant answer. This feature sets the proposed model apart from the commonly used information retrieval and extraction based techniques.

The Panchatantra collection[2] comprising of 65 short stories was used to experiment on the model. This dataset had variety of stories with different lengths. The questions on each of these stories were framed by multiple annotators and best of which were picked to validate the system. The answers given by the annotators were used as gold data to measure the quality of the system.

2 Architecture and Design

The Samajh-Boojh System is broadly classified into two parts: comprehension part and question answering part. The system works similar to human approach of answering reading comprehension questions. The system

*equal contribution

[1]Samajh-Understanding and Boojh-Analysis, which translates to reading comprehension in Hindi.

[2]https://www.hindisahityadarpan.in/2016/06/panchatantra-complete-stories-hindi.html

D M Sharma, P Bhattacharyya and R Sangal. Proc. of the 16th Intl. Conference on Natural Language Processing, pages 239–248
Hyderabad, India, December 2019. ©2019 NLP Association of India (NLPAI)

takes passage and queries corresponding to the passage as the input, and returns answers to the questions. In subsequent sections we dwell deeper into these parts and see how they function.

2.1 Reading Comprehension Part

The reading comprehension part of the system is responsible for structuring the story into a **Discourse Collection** which contains the characteristics of the story. This structure is inspired by Thorndyke's Story Grammar (Thorndyke, 1977). Discourse Collection is comprised of the following components:

Episode_Id: Unique key associated to the sentence. Its incrementally assigned as we parse the sentences. In Discourse Collection, we associate each logical sentence as an episode.

Original_sentence: The WX version of the logical sentence found in the story.

Time: The time setting in which the episode took place. If the Original_Sentence doesn't specify the time, this field is populated from the previous episode's Time value. Default is 'tbd: to be decided'

Location: The place in which the episode took place. If the Original_Sentence doesn't specify the location, this field is populated from the previous episode's Location value. Default is 'tbd: to be decided'

Karta: The karta (doer) of the logical sentence is given as the value. This is obtained from the dependency parser[3].

Karta_Adpos: The Adpos (adjective and prepositions) associated with the Karta to frame the answers during the question-answering stage.

Karma: The karma of the logical sentence is populated in this column.

Karma_Adpos: The Adpos (adjective and prepositions) associated with the Karma to frame the answers during the question-answering stage.

Anaphora_Resolved_Sentences: The logical sentence in which the anaphora is replaced with noun.

Root_Node_Sentences: The words of the logical sentences are replaced with their roots. For this we used the shallow parser [4].

Given: The sentence which is related to the current sentence.

New: The current sentence which is having the Given sentence as a prerequisite.

Parser_Output: The output of the dependency parser.

The overview of the Reading Comprehension system is seen in Figure 1. The passage is given as the input to **Logical Sentence Module** to break it into logical sentences, the split passage is given as input to the Discourse Generator module which contains **Graph Maker Module**, **Anaphora Resolution Module**, **Root Node Resolution Module** and **Discourse Information Filler Module**. The final output of these four modules is the Discourse Collection which is the output of comprehension system. The individual modules of the reading comprehension system along with detailed working is explained in the forthcoming sections.

Logical Sentences Module

Even though the passage can easily be split into words, sentences and paragraphs when given as the input, it's a challenge to extract the semantics. We break the sentences based on the generic punctuation marks such as full stop, comma, semicolon, question mark, exclamation mark and conjunctions such as और, कि, पर, कर, फिर, इसीलिए, तब, तो, क्योंकि, क्यूंकि, लेकिन, परंतु, किन्तु.

When splitting the sentences by the split words, we noticed the tags were improper at some instances. Example:
S1:राम घर जाना चाहता था पर नहीं जा पाया।
T1: Ram wanted to go home but couldn't go.
S2:राम घोड़े पर बैठा था।
T2: Ram sat on the horse.

Here the word पर translates to 'but' and 'on', we want to split the sentence into two parts only in S1 and not in S2. The POS tags using Dependency Parser[3] give 'PSP' tag, hence making it difficult to differentiate on which पर to split the sentences. To resolve this issue we decided to see the context of the given split word and then make the decision. The logic for deciding whether to split or not is given

[3]https://bitbucket.org/iscnlp/parser/src/master/README.rst

[4]http://ltrc.iiit.ac.in/analyzer/hindi/index.cgi

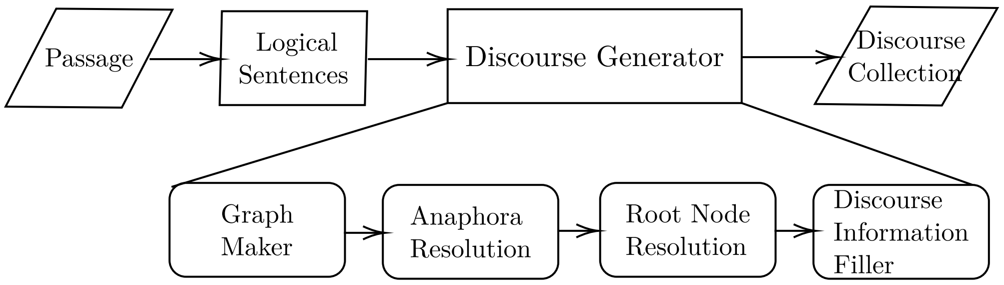

Figure 1: Comprehension Design

below:
If the word before the split word is verb ie. having POS tag as 'VM' or 'VAUX' then we split the sentence into two parts and populate the Discourse Collection with two episodes each containing the split sentences as Original_Sentences(OS).
In the Figure 2, we see in the sample passage that apart from the full stops, the sentences are split at 'taba/तब' and 'para/पर' resulting in 5 episodes.

Graph Maker

The Original_Sentence which was populated in Logical Sentence Module is sent through the dependency parser [3] and the parser output is stored in the Discourse Collection of the corresponding episode. From the parser output, if there exits any **k7t** relation, it is stored in the 'Time' slot of the episode. If there exists a **k7p** relation, it is stored in the 'Location' slot of the episode. The word with **k1** relation is stored in the 'Karta' slot of the episode along with the 'lwg___psp' as case marker of the Karta and word with relation 'nmod___adj' as the Karta adjective, the case marker and adjective with Karta word are called Karta_adpos and stored in the corresponding episode. The word with **k2** relation is stored in Karma slot of the episode. The child nodes of the Karma in dependency tree who have the relations 'lwg___psp' and 'nmod___adj' are stored as Karma_Adpos.

Anaphora Resolution Module

We use the Original_Sentence from the episode to resolve the anaphora and store it in Anaphora_Resolved_Sentence(ARS) of the corresponding episode. We used the algorithm mentioned in Dakwale(2014) to resolve the anaphora. This algorithm is a right fit as it uses rules from the dependency parser[3].
We see in Figure 2 that the the word 'vaha/वह' translates into rAma in episode 2 and billI in episode 4 based on the context, 'usE/उसे' and 'vO/वो' are resolved into rAma in episode 2 and 5.

Root Node Resolution Module

The root of a word is important when we are comparing two sentences. We convert each word in Anaphora_Resolved_Sentence of the episode into its root form and store it as Root_Node_Sentence(RNS) for corresponding episode. We used the IIIT Parser[4] and the output was parsed through the SSF format mentioned in Bharati(2007) and the root form of the words were extracted.
In episode 2 the word 'gaya' changes to 'ja' in Figure 2.

Discourse Information Filler

Discourse is when we look beyond the scope of a sentence and use information between their relation. Here we fill the 'Given' and 'New' values of the episode. The default values are 'tbd: to be decided'. If the passage sentence has split words mentioned in section 2.1, the sentence is split into two episodes such that, the second episode will contain the first split sentence as 'Given' in its Discourse Collection and second split sentence as the 'New'. Our assumption and complexity is limited to identifying a co-dependency between two sentences if they are separated by split words.
The output of this stage will give the Discourse Collection. Episode 2 in the Figure 2, has 'Given' and 'New' values populated since it has the word 'taba/तब' (translation: then).

Passage:

rAma Eka acchA ladkA thA. vaha Eka dina pAThaSAlA jA rahA thA taba usE Eka billI dikhI. rAma usakE pAsa gayA para vaha bhAga gaI aura vO dukhI hO gayA

Discourse Collection:

```
{
”0”: {
       ”OS”: ”rAma Eka acchA ladkA thA”,
       ”karta”: ”rAma”,
       ”kartaadj”: [”rAma”, ”acchA”],
       ”ARS”: ”rAma Eka acchA ladkA thA”,
       ”RNS”: ”rAma Eka acchA ladka thA”
         }
”1”: {
       ”OS”: ” vaha Eka dina pAThaSAlA jA
                rahA thA”,
       ”time”: ”din”,
       ”location”: ”pAThaSAlA”,
       ”karta”: ”rAma”,
       ”ARS”: ”rAma Eka dina pAThaSAlA jA
                 rahA thA”,
        ”RNS”: ”rAma Eka dina pAThaSAlA jA
                 rahA thA”
        }
”2”: {
       ”OS”: ”usE Eka billI dikhI”,
       ”time”: ”din”,
       ”location”: ” pAThaSAlA”,
       ”karta”: ”billI”,
       ”given”: ” rAma Eka dina pAThaSAlA jA
                  rahA thA”,
       ”new”: ”rAma Eka billI dikhI”,
       ”ARS”: ”rAma Eka billI dikhI”,
       ”RNS”: ”rAma Eka billI dikha”
        }
”3”: {
       ”OS”: ” rAma billI pAsa gayA”,
       ”time”: ”din”,
       ”location”: ” billI pAsa”,
       ”karta”:”rAma”,
       ”ARS”: ”rAma billI pAsa gayA”,
       ”RNS”: ”rAma billI pAsa jA”
        }
”4”: {
       ”OS”: ”vaha bhAga gaI”,
       ”time”: ”din”,
       ”location”: ” billI pAsa”,
       ”karta”: ”rAma”,
       ”given”: ” rAma billI pAsa gayA”,
       ”new”: ”billI bhAga gaI”,
       ”ARS”: ”billI bhAga gaI”,
       ”RNS”: ”billI bhAga jA”
        }
”5”: {
       ”OS”: ”vO dukhI hO gayA”,
       ”time”: ”din”,
       ”location”: ” billI pAsa”,
       ”karta”: ”rAma”,
       ”given”: ”billI bhAga gaI”,
       ”new”: ”rAma dukhI hO gayA”,
       ”ARS”: ”rAma dukhI hO gayA”,
       ”RNS”: ”rAma dukhI hO jA”
        }
}
```

Figure 2: Discourse Collection
The passage and its corresponding discourse collection is shown here. Only the populated values are shown, rest all are ‘tbd-to be decided’ except for parser_output. OS-original_sentence, ARS-Anaphora_Resolved_Sentence, RNS-Root_Node_Sentence

Question Type	Question Words
Karta	‘kisnE’ ‘kisakE’ ‘kauna’ ‘kisasE’
Karma	‘kisakO’ ‘kisakI’
Time	‘kaba’ ‘samaya’ ‘dina’
Loc	‘kahA ’
Recipient	‘kisE’
Adj_Noun	‘kaisA’ ‘kaisI’
Intf	‘kitnA’ ‘kitnE’
Kya	‘kyA’
Kiske	‘kiskE’
Kiska	‘kiskA’ ‘kiskI’
GivenNew	‘kara’ ‘bAda’ ‘phalE’ ‘kyoM’

Table 1: Types of Questions.

2.2 Question Answering Part

The Question-Answering part of this model takes Discourse Collection which was output of the Reading Comprehension part, as the input along with the query related to the passage and returns the answer as the output. The brief overview of the system is shown in Figure 3. The query is fed in the Devanagari format and the answer is given in the same. The working of this system is seen in following sections.

Question Analyzer Module

This module takes the Devanagari format of the input query and returns the type of the question along with key word relevant to the query. This format is similar to that found in QLL (Vargas-Vera et al., 2003).

We tag the questions based on the question words into 11 major categories shown in Table 1. This list can be expanded as per the required answers from the question word. For example,
Q1: गांव में कितने मुर्गे रहते थे?
T1: How many chickens were there in the village?
The answer expects the quantity of the chickens. So, we place it into ‘Intf’ category

We now see each type of question in detail and see how they are handled:
Karta: It involves the question words which requires the answer as the doer. किसने बंदर को

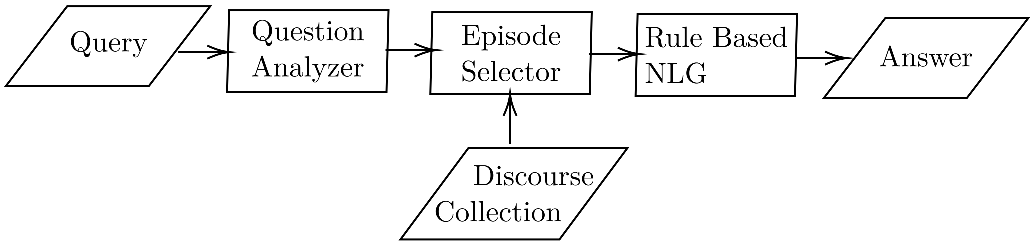

Figure 3: Question Answering Design

परेशान किया? (Who troubled the monkey?)
Karma: The question which requires the answer as the act of the sentence. बल्ली किसको दिखी? (Who saw the cat?)
Time: The question which expects the answer as the time. सूरज कब घर आया? (When did Suraj come home?)
Loc: The question word which requires answers as the location. किताब कहाँ राखी थी? (Where was the book kept?)
Recipient: The question which expects the answer of the receiver.किसे चोट लगी थी? (Who got injured?)
Adj_Noun: The question which requires the answer as the adjective of particular noun.राम कैसा लड़का था? (What kind of boy was Ram?)
Intf: Question which requires the answer as the quantity of a particular entity. एक गांव में कितने लोग थे? (How many people stayed in the village?)
Kya: When the answer is supposed to describe or explain the situation. श्याम ने खाने में क्या खाया? (What did Shyam have for his meal?)
Kiske: When the question requires the entities involved with mentioned subject as the answer. सीता किसके साथ खेल रही थी? (With whom was Sita playing?)
Kiska: The question is seeking the possessive trait of the entity mentioned. यह किताब किसकी है? (Whose book is this?)
GivenNew: The question that gives an activity and requests for the *consequence* of the activity falls in category of GivenKnown. The questions which describe an activity and expects the *cause* of the activity as the output, it falls in category of NewKnown.
Example:
कुछ लड़कों ने एक बिल्ली को तंग किया और वह परेशान हो गयी (Some boys troubled a cat and the cat got angry)
Given Known:
बिल्ली तंग होने के बाद क्या हुआ? (What Happened after the cat was troubled?), Given Info from the question: बिल्ली तंग हुई(Cat was troubled) Here, the answer is expected to be the consequences of cat being troubled.
New Known:
बिल्ली परेशान क्यों हो गयी? (Why was the cat angry?), new info from the question: बिल्ली परेशान हुई (cat is angry) Here, the answer is expected to address the reasons why the cat was angry. The output of the question analyzer module for above mentioned question types is shown in Figure 2.

Question Type	**Output Format**
Karta	[‘Karta’]
Karma	[‘Karma’]
Time	[‘Time’]
Loc	[‘Loc’]
Recipient	[‘Recipient’]
Adj_Noun	[‘Adj_Noun’, one word before the question word]
Intf	[‘Intf’, one word after the question word]
Kya	[‘Kya’]
Kiske	[‘Kiske’, one word after the question word]
Kiska	[‘Kiska’]
GivenNew	[‘GivenNew’, the information which is either new or given]

Table 2: Output of the Question Analyzer.

There is a preference order given to these question types, in cases of when two words belonging to two different classes (in Table 1) exist in same query. The observed overlaps include: Time and Kya: in this case the question type will be treated as Kya. Kya and Given-New: in case of this overlap, the question type will be treated as Kya.

Episode Selector Module

The input to this module is the output of the Query Analyzer module (Table 2) and Discourse Collection(Figure 2) which is the output of the Reading Comprehension stage of our system. This is seen clearly in Figure 3. The episode is detected by using Jaccard similarity between the Query(Q) and Root_Node_Sentence(RNS) whose formula is as follows:

$$Jaccard\_Sim(Q, RNS) = \frac{n(Q \cap RNS)}{n(Q \cup RNS)}$$

Where, $n(Q \cap RNS)$ is number of common words in the Query and Root_Node_Sentence and $n(Q \cup RNS)$ is the total number of words in Query and Root_Node_Sentence.

Algorithm 1 Weighted Jaccard Similarity

```
procedure JACCARDSIM($Q, RNS$)
    $NounTags \leftarrow \{MNP, MNS, NN\}$
    $VerbTags \leftarrow \{VM\}$
    $VAuxTags \leftarrow \{VAUX\}$
    $AdTags \leftarrow \{JJ, RB\}$
    $Union \leftarrow Q \cup RNS$
    $Intersec \leftarrow Q \cap RNS$
    $JS \leftarrow 0$
    for all $word \in Intersec$ do
        switch $POS(word)$ do
            case $\in AdTags$
                $JS \leftarrow JS + 5$
            case $\in VerbTagss$
                $JS \leftarrow JS + 4$
            case $\in NounTags$
                $JS \leftarrow JS + 3$
            case $\in VAuxTags$
                $JS \leftarrow JS + 2$
            case $default$
                $JS \leftarrow JS + 1$
    end for
    $NormalisedJS \leftarrow JS/|Union|$
    return $JS, NormalisedJS$
end procedure
```

We have modified the formula to give better results. The formula is our version of weighted Jaccard similarity, wherein, we take the POS tags of the words which are common between the Query and the Root_Node_Sentence, and give more weightage to the word if it is less frequent, rather than focusing on frequently occurring words, which don't capture the similarity between the Query and Root_Node_Sentence such as prepositions. We have given the priority to rare words based on their POS word tags. Priorities of POS tags as given as Adjective/Adverb > Verb > Noun > Auxillary Verb > Others.
The respective POS tags from the parser are: $(JJ/RB) > (VM) > (NN/NNS/NNP) > (VAUX) > Others$. We call this as the Jaccard_Score between the episode and the Query, if we divide Jaccard_Score by $(Q \cap RNS)$, we get Normalized_Jaccard_Score.
After calculating the Jaccard_Score and the Normalized_Jaccard_Score for each episode in Discourse Collection, we take the episode which has the highest Jaccard_Score, if two episodes have highest Jaccard_Score, we compare their Normalized_Jaccard_Score and choose the higher valued episode as our chosen episode. The pseudo code is given in Algorithm 1.
The Output of this module is the Episode_Id(from the Discourse Collection) which has maximum similarity to the Query.

Rule Based Natural Language Generator Module

This Module generates answer for a given query. It takes the episode chosen by the Episode Selector, Discourse Collection, and the query as input and generates the answer according to the query type.
Answer for various question types(Table 2) is generated as follows:
Karta: We extract the Karta from the Discourse_Collection for the chosen episode along with Karta_Adpos(Karta_Adjective and Karta_Case_Markers) and give the answer as Karta_Adjective + Karta + Karta_Case_Marker. The answer to the question kIsnE zyAma kO kalama dI? [Karta Question Type] answer will be rAma nE (refer Figure 4).
Karma: We extract the Karma from the

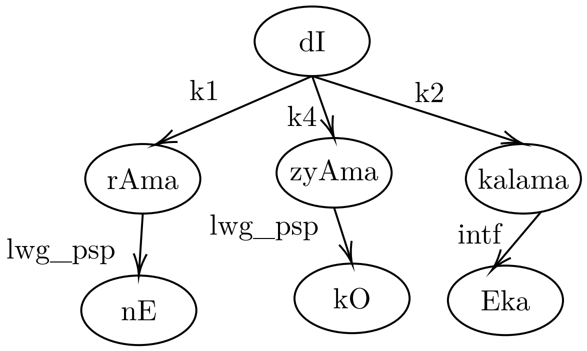

Figure 4: Karta, Recipient, Intf and Kya

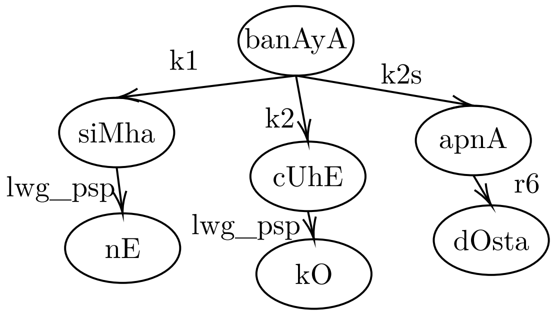

Figure 5: Karma

Discourse_Collection for the chosen episode along with Karma_Adpos(Karma_Adjective and Karma_Case_Markers) and give the answer as Karma_Adjective + Karma + Karma_Case_Marker. If the 'Karma' slot of the Discourse_Collection isn't populated, the Anaphora_Resolved_Sentence of the chosen episode is returned. The answer to the question siMha ne kiskO apnA dosta banAyA? [Karma Question Type] answer will be cUhE kO (refer Figure 5).

Time: If the Parser_Output of the given episode contains the 'k7t' relation between two nodes, then the child node is the output. If there doesn't exist any 'k7t' relation, then 'k7' relation is used and the child node is the answer. If either of these aren't existing, then time is extracted from the 'Time' slot of the Discourse_Collection of the given episode and is displayed as the answer. The answer to the question sIta kaba pathzAlA gayI? [Time Question Type] answer will be subah (refer Figure 6).

Loc: If the Parser_Output of the given episode contains the 'k7p' relation between two nodes, then the child node is the output. If there doesn't exist any 'k7p' relation, then 'k7' relation is used and the child node is the

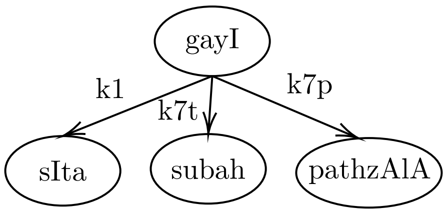

Figure 6: Loc and Time

answer. If either of these aren't existing, then location is extracted from the 'Loc' slot of the Discourse_Collection of the given episode and is the answer. The answer to the question sIta kaha gayI? [Loc Question Type] answer will be payhzAla (refer Figure 6).

Recipient: The main verb(MV) is extracted from the Parser_Output of the Episode. If the MV shares relation 'k4' with a child we return that child as the answer. If there doesn't exist any child with 'K4' relation, we check for any child nodes of the MV with 'k4a' relation and return that as the answer. The answer to the question rAma ne kisE kalama dI? [Recipient Question Type] answer will be zyAma (refer Figure 4).

Adj_Noun: From Table 2, we can see that the output is the Noun whose Adjective is asked in the question. Let the Noun whose adjective is asked be MN. We take the Parser_Output of the chosen episode and check for the child nodes of the MN who have relation 'nmod___adj' and return the child node as the answer. The answer to the question kalama kaisI hai? [Adj_Noun Question Type] answer will be suNdara (refer Figure 8).

Intf: From the 2, we can see that the Noun(MN) whose quantity is asked is returned along with the classification of the question. To get the answer we take the Parser_Output of the chosen episode and search for the MN, then we check for child nodes of MN who have relation 'intf' with MN, lets call the child 'NounIntf', we then check child nodes who have relation 'nmod___adj' with MN, lets call it 'NounAdj'. If 'NounIntf' and 'NounAdj' exist, we return the answer as NounIntf + NounAdj and MN as the answer. If 'NounAdj' doesn't exist, we just return NounIntf + MN as the answer. The answer to the question rAma ne kitnE kalama dIyE? [Intf Question Type] answer will be Eka (refer Figure 4).

Kya: We extract the Main Verb (MV) from

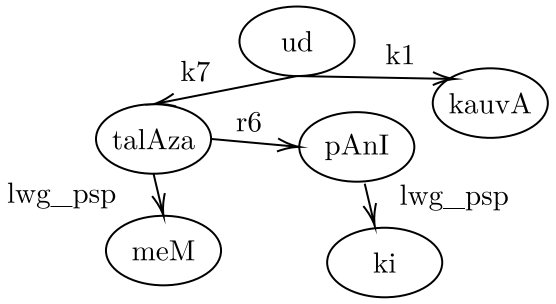

Figure 7: Kiske

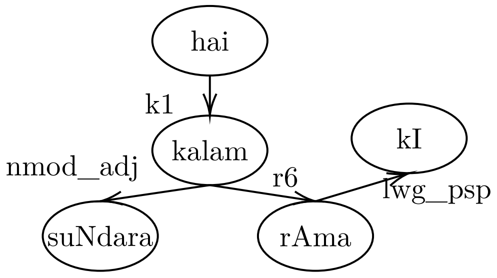

Figure 8: Kiska, Adj_Noun

the Parser_Output of the Discourse Collection of the chosen episode. We then check if either of 'k1s', 'pof', 'k2' relations exist between the MV and its children. If it exists, we check if the child is mentioned in the question, if it isn't mentioned in the question, we return child Node as the answer. If no child exists with above mentioned relations or if it exists and the child has occurred in the question itself then, we check if the 'Given' slot of the discourse Collection is populated for the next episode and return the value in 'NEW' slot of the Discourse Collection as the answer. The answer to the question rAmA nE kyA dIyA? [Kya Question Type] answer will be kalama (refer Figure 4).
Kiske: From Table 2, we observe that subject whose entity is asked is known. We consider this subject as Main Noun(MN), we check the children of MN who have relation 'k7' and return it as the answer. If this relation doesn't exist, we check for children with 'r6' relation and return it as the answer. 'r6' and 'k7' are called *SambandhRelations* in Panninian Grammar (Bharati et al., 1995). The answer to the question kauvA kiske talAza meM udA? [Kiske Question Type] answer will be pAnI (refer Figure 7).
Kiska: We extract the Main Verb(MV) from the Parser_Output of the Discourse Collection for the particular episode. We check the children of MV which have relations in order, 'k2', 'k7', 'r6'. The answer to the question yaha kalama kiskI hai? [Kiska Question Type] answer will be rAma (refer Figure 8).
GivenNew: We can see from Table 2, the information of GivenNew is given as the output of the Query Analyzer step. If the question is 'Given Known' (Refer 2.2), we choose the episode based on the highest Jaccard similarity (mentioned in 2.2) between the 'Given' slot of the Discourse Collection and the query. The Episode which gets the highest score, is the chosen episode and we return the 'New' slot value as the answer. Similarly for the 'New Known' (Refer 2.2) we choose the episode based on the 'New' slot and the answer is in 'Given' slot of the same episode. The answer to the question jaba rAma billI kE pAsa gayA taba kyA huA? [GivenKnown Question Type] the answer is billI bhAga gaI. (refer Figure 2)
Default: In case an unknown question type is encountered or no answer has been given for the above question types, we return the Anaphora_Resolved_Sentence of the chosen episode as the answer.

3 Experiments

Panchatantra is a collection of fables. It has five parts, Mitra-Bheda (The loss of friends), Mitra-laabha (The winning of friends), Kakolukiyam (on crows and owls), Labdhapranasam(Losing what you have gained) and Apariksitakarakam (Ill-Considered actions). We have chosen a corpus of 65 stories from the tales across all parts of Panchantantra to test our system.

We collected the stories from the link mentioned in footnote[2] and fixed the syntax (punctuation and spellings). We annotated 440 questions for the above stories and assigned the question types, the ideal episode to be selected from the Discourse Collection and the correct answer for each of the questions. While annotating the questions, we made a conscious effort to involve more questions in type 'Kya' and 'GivenNew' (refer Table 4), since they are versatile concepts and we intended to test them extensively against the rules we formatted as the other types are more or less intuitive on the dependency parser tags[3]. We randomly

Story	Original Sentence	Anaphora Resolved Sentence	Root Node Sentence
1	3/7	5/7	5/7
2	7/11	7/11	11/11
3	3/6	3/6	4/6
4	3/5	4/5	4/5
5	1/8	3/8	4/8
6	1/3	2/3	2/3

Table 3: Episode selection accuracy

selected 10 stories from the corpus along with the questions and generated rules based on linguistic heuristics, to avoid overfitting. We then verified the rules on the remaining stories.
Out of 440 questions, 72 more questions were rightly answered on using weighted Jaccard similarity when compared to normal Jaccard similarity. That is, *16% episodes were rightly selected when weighted Jaccard similarity was used instead of normal Jaccard similarity on our model.*
We also compared episode selection accuracy by including modules mentioned in figure 1 on 6 random stories which are different from the previously selected stories. The results can be seen in table 3 for the same. It can be observed that the weighted Jaccard similarity accuracy improved consistently as we added the Root Node Resolution and then the Anaphora Resolution modules. The below example demonstrates the improvement in episode selection accuracy for different modules:
Question: राम क्या खा रहा था? [Translation: What was Ram eating?]
Original Sentence: उसने आम खाया. [Translation: He ate a mango] **Jaccard Score: 0**
Root Node Sentence: वह आम खा [Translation: He eat mango] **Jaccard Score: 4**
Anaphora Resolved Sentence: राम आम खा[Translation: Ram eat mango] **Jaccard Score: 7**
Overall accuracy of the answers based on question Types is Shown in Table 4. The figure clearly shows good accuracy for majority of the questions. Since the 'Kya' and 'GivenNew' format of the questions are versatile and the answers can be subjective, the

Question Type	Questions	Accuracy
Karta	35	94.3%
Karma	7	100%
Time	7	100%
Loc	45	100%
Recipient	15	100%
Adj_Noun	15	100%
Intf	15	100%
Kya	179	71.6%
Kiske	13	84.62%
Kiska	5	100%
GivenNew	33	63.7%
Total	**440**	**75.45%**

Table 4: Accuracy of the answers

accuracy for these categories cannot be comparable to the direct question types whose answers are obtainable through dependency parser tags solely. Overall accuracy of the system is *75.45%*.

4 Future Work

This model currently doesn't answer कैसे [How] types of questions, which can be included in future.
Currently we don't resolve synonyms and antonyms to answer the questions, which when done, can improve Episode Selection algorithm and also aim at answering complex questions. The current model assumes the passage to be in chronological order. We can improve the model if we capture the relative time of the episode to suite the passages which don't follow the chronological order.
Versatile questions such as 'GivenNew' and 'Kya' can be improved by increasing the scope of answer retrieval to multiple sentences or episodes around the selected episode unlike the single episode range implemented in our model. This will in-turn also increase the scope of model to include longer and complex texts.

5 Conclusion

Reading Comprehension is a complex task, which involves comprehending the passage and answering the questions following the passage. Once, we are able to structure this unstructured data(passage), we can answer the questions relatively well without complex approaches. The rules in linguistic are intuitive and are capable of answering complex questions. Since it's rule based, there is no requirement of large data to obtain promising results. In the model we managed to get 75% accuracy with just 65 stories and managed to answer wide range of answers. This model is versatile and can be extended to other Indian languages provided the dependency parser (similar to one we used[3]) exists.

References

Akshar Bharati, Vineet Chaitanya, Rajeev Sangal, and KV Ramakrishnamacharyulu. 1995. *Natural language processing: a Paninian perspective.* Prentice-Hall of India New Delhi.

Akshar Bharati, Rajeev Sangal, and Dipti M Sharma. 2007. Ssf: Shakti standard format guide.

Praveen Dakwale. 2014. *Anaphora Resolution in Hindi.* Ph.D. thesis, PhD thesis, International Institute of Information Technology Hyderabad.

Shriya Sahu, Nandkishor Vasnik, and Devshri Roy. 2012. Prashnottar: a hindi question answering system. *International Journal of Computer Science & Information Technology*, 4(2):149.

Perry W. Thorndyke. 1977. Cognitive structures in comprehension and memory of narrative discourse. *Cognitive Psychology*, 9(1):77 – 110.

Dr. Maria Vargas-Vera, Enrico Motta, and John Domingue. 2003. Aqua: an ontology driven question answering system.